ЧЕЛОВЕК И КОМПЬЮТЕР

В.Ф.Очков
Ю.В.Пухначёв

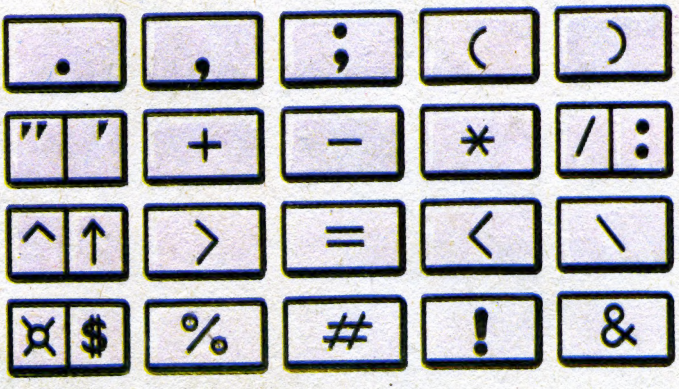
24 ЭТЮДА НА БЕЙСИКЕ

Авторы книги — преподаватели высшей школы (В.Очков — МЭИ, Ю.Пухначев — МФТИ) попытались в программных этюдах раскрыть лишь одну из сторон работы с персональным компьютером — составление программ на Бейсике.

СТАНДАРТНЫЕ ФУНКЦИИ БЕЙСИКА

Встроенные функции	Их аналоги в традиционной символике
SQR (X)	$\sqrt{x}$
EXP (X)	e^x
LOG (X)	ln x
LOG10 (X)	log x
ABS (X)	|x|
SIN (X)	sin x
COS (X)	cos x
ATN (X)	arctg x
INT (X)	[x]
SGN (X)	sign x
PI (X)	πx
RND (X)	случайное число

СИМВОЛЫ БЕЙСИКА



Комбинации символов

В.Ф.Очков
Ю.В.Пухначёв

24 ЭТЮДА НА БЕЙСИКЕ

МОСКВА
"ФИНАНСЫ И СТАТИСТИКА"
1988

ББК 22.18
О-95
УДК 681.3.06

Рецензенты: А. Б. БОЙКО, С. В. ЧЕРЕМНЫХ

Очков В. Ф., Пухначёв Ю. В.

О-95 24 этюда на Бейсике. — М.: Финансы и статистика, 1988. — 175 с.: ил. — (Человек и компьютер).
ISBN 5 — 279 — 00108 — 2

Как алгоритмизировать задачу, написать программу, реализующую данный алгоритм, как ввести ее в компьютер, как выполнять расчет — все эти этапы описаны в книге в виде диалогов, участники которых обсуждают решение практических задач из области математики, информатики, научной организации труда, а также игровые и развлекательные программы.

Для инженерно-технических работников, учителей и других читателей — будущих пользователей персональных ЭВМ.

О $\frac{2405000000 —}{010(01) — 88}$ 123 — 88 ББК 22.18

ISBN 5 — 279 — 00108 — 2

© Издательство «Финансы и статистика», 1988

ПРЕДИСЛОВИЕ

Книга «24 этюда на Бейсике» адресована читателям, желающим получить представление о работе на персональных компьютерах.

При освоении новой техники особое значение имеет начальный этап привыкания к ней. Он еще не приносит устойчивых навыков работы — да это от него и не требуется; он должен лишь вселить уверенность, что эти навыки вполне достижимы, причем наиболее легко и быстро — благодаря использованию множества не сложных, но эффективных приемов, своеобразных «маленьких хитростей». Этот этап можно пройти, располагая довольно простым образцом новой техники и даже не работая самому, а наблюдая за чьими-то уверенными действиями.

Именно такой путь предлагают своему читателю авторы книги. Каждый из собранных в ней этюдов — это живая сценка, сюжетным центром которой служит та или иная задача, решаемая с помощью персонального компьютера. В рассказах и беседах участников сценки либо комментируется уже готовая программа, либо описывается процесс ее составления. По ходу диалогов читатель знакомится с приемами программирования и работы на персональных компьютерах.

В чередовании участников сцен нет никакой логической закономерности. Начальника лаборатории сменяет хозяйка дома, где недавно появился бытовой компьютер; инженера-программиста — поэт, заинтересовавшийся столь модной сегодня компьютерной премудростью. Вместе с тем довольно строгой закономерности подчинено содержание их разговоров: раскрываемые в них приемы программирования усложняются от этюда к этюду, все более пространными и логически сложными становятся обсуждаемые программы.

Впрочем, такая последовательность отнюдь не делает книгу учебным пособием. Создаваемое ею представление о работе на персональных компьютерах не может не быть в известной степени общим и односторонним. Общность обусловлена небольшим объемом книги; односторонность — тем, что все приведенные в книге программы составлены только на Бейсике, получившем сегодня наибольшее распространение в мире персональных компьютеров.

Машины этого класса позволяют создавать довольно емкие базы данных, проводить оперативные расчеты, редактировать тексты, представлять результаты работы в удобном виде (графики, таблицы и т. д.). Сравнивая используемые для этого

языки программирования, можно заметить, что каждый из них при всей широте диапазона своих применений имеет на нем какие-то особенно выигрышные участки. Этим объясняется предпочтение, которое отдают тому или иному языку пользователи. Не последнюю роль при выборе языка играет его простота.

Удачное сочетание универсальности и простоты принесло Бейсику широкую популярность, отмечавшуюся выше. Программы, написанные на нем для различных машин, как правило, обнаруживают несходство в некоторых деталях. Поэтому принято говорить о различных версиях, или диалектах Бейсика. Наиболее богатые их них предоставляют хорошие возможности для расчетов и обработки текстов, более скромные — для создания графиков и работы с таблицами. Весь спектр таких возможностей, конечно, не нашел отражения в этой небольшой книге. Здесь для написания программ привлекаются средства, общие для большинства версий Бейсика. Это сделано для того, чтобы читатель смог продолжить знакомство с языком, работая на различных машинах. Если в разных версиях какие-то равноценные программные средства выражаются в неодинаковой форме, то авторы в таких случаях, как правило, придерживались символики, принятой на компьютере «Искра 226» и лишь в отдельных программах показывали, как то же самое можно выразить в виде, «понятном» для компьютеров СМ-4, ИБМ ПС и т. д.

Книга состоит из четырех глав. Программы, вошедшие в первую главу, несложны: в них используются лишь простые числовые переменные. Здесь читатель узнает о наиболее употребительных операторах Бейсика. В программах второй главы наряду с простыми числовыми встречаются также индексные переменные. С литерными переменными, как простыми, так и индексными, читатель знакомится в третьей главе. Программы четвертой главы нуждаются для своего выполнения во внешних носителях информации — магнитных лентах, магнитных дисках.

В конце каждого этюда помещено два-три коротких комментария авторов к существенным особенностям только что разобранной программы или задача, предлагаемая читателю для самостоятельного решения.

Эти комментарии можно подразделить на пять групп:

— основные понятия программирования;

— принципы составления программ, приемы работы на персональных компьютерах;

— символика Бейсика и ее варианты в различных версиях языка;

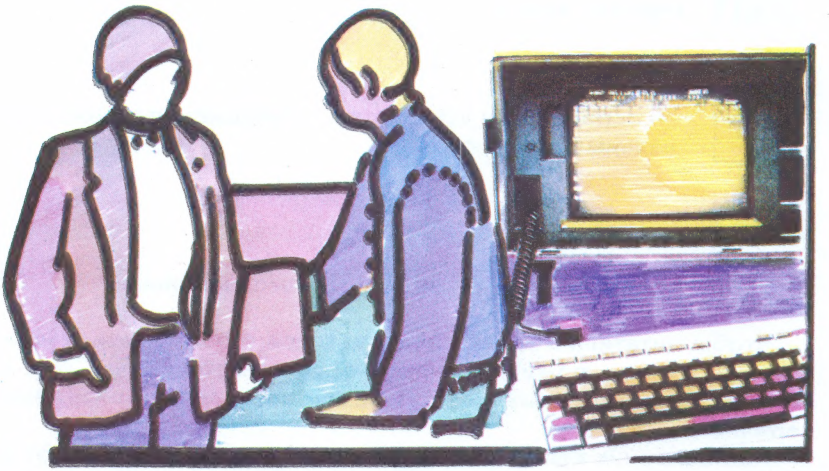
— особенности устройства и функционирования персональных компьютеров;

— задачи для самостоятельного решения.

Чтобы читатель легче ориентировался в этих комментариях, каждый из них помечен значком, соответствующим той или иной из перечисленных групп, как это показано в приведенном перечне.

На последних страницах книги читатель найдет список встречающихся в ней операторов Бейсика (причем каждый дан в нескольких вариантах, употребляемых в различных версиях языка), небольшой словарь программистских терминов и предметный указатель.

В заключение несколько слов по поводу термина «этюд», стоящего в заголовке книги. На языке музыкантов и шахматистов он означает небольшое интересное упражнение для отработки того или иного элемента игры. Следуя такому толкованию, авторы старались подобрать для иллюстрации каждого аспекта программирования задачу позанимательнее и в то же время такую, чтобы для ее решения не требовалось ни сложных логических построений, ни пространных математических выкладок. У художников слово «этюд» понимается как зарисовка, выполненная с натуры и представляющая собой часть будущей большой картины. Авторы надеются, что у читателя, внимательно следившего за решением задач, сложится довольно ясная картина работы на персональных компьютерах, а разбор программных этюдов пригодится при написании больших программных полотен.



Этюд 1. УГАДАЙ ЧИСЛО

Я задумал написать книгу о персональных компьютерах, об особенностях работы на этих машинах, о составлении программ для них на языке Бейсик. В издательстве мою заявку одобрили и уже через несколько дней познакомили с художником-иллюстратором будущей книги.

Художник, желая сделать ее оформление как можно более соответствующим содержанию, попросил показать ему персональную ЭВМ, объяснить принцип ее действия. Я пригласил его к себе домой, полагая, что мой персональный компьютер при всех его скромных возможностях вполне годится для первого знакомства с вычислительными машинами этого типа.

Художнику не терпелось разобрать решение какой-нибудь простенькой арифметической задачи. Но я убедил его в том, что современные ЭВМ используются не только для вычислений, а потому мы сможем ознакомиться с работой компьютеров на более понятных примерах, например сыграв с машиной в игру «Угадай число».

Сев за *дисплей* (рис. 1.1), я набрал на *клавиатуре* (рис. 1.2) программу игры и объяснил ее правила. Они предельно просты. ЭВМ задумывает некоторое четырехзначное число. Его надо отгадать за минимум попыток.

Я нажал пусковую клавишу ВК. На дисплее появился вопрос: «1-я попытка?».

— В ответ на вопрос машины, — продолжил я объяснение, — игрок должен набрать на цифровых клавишах пробное число.

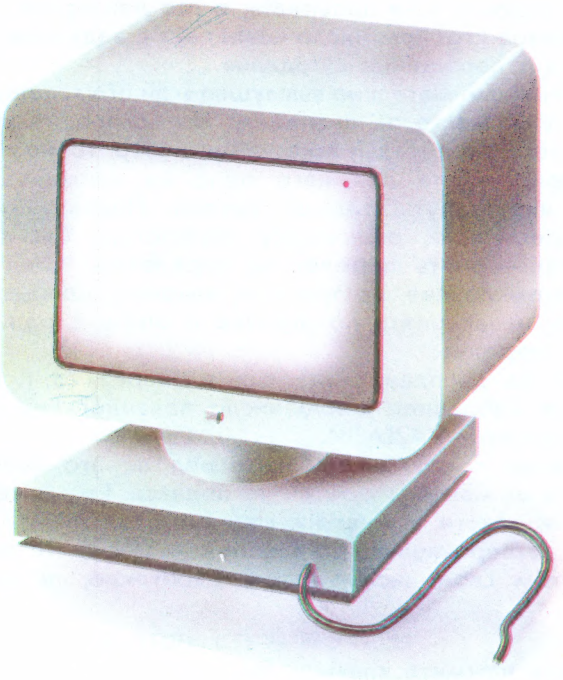


Рис. 1.1. Дисплей ЭВМ

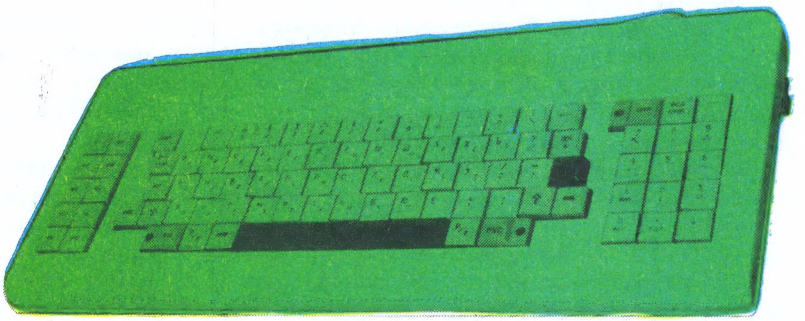


Рис. 1.2. Клавиатура дисплея

Оно тоже появляется на дисплее, строчкой ниже. Вслед за этим компьютер выведет на дисплей слово «недолет» или «перелет», судя по тому, меньше или больше набранное игроком число по сравнению с задуманным машиной. Потом на дисплее по-

является надпись «2-я попытка?», и все повторяется вновь. Если же введенное игроком число совпадает с задуманным машиной, она выводит сообщение «угадали».

Художник внимательно выслушал мои объяснения и попросил уступить ему место за дисплеем. Не помню, сколько попыток ушло у него на отгадывание. Много, это точно. Машина, как назло, приготовила для него одно из наименьших четырехзначных чисел, чуть больше тысячи. Первая попытка моего гостя, естественно, окончилась перелетом. Он стал сбавлять по десятке — опять перелет за перелетом. Стал сбавлять по сотне, потом по две, по три — и, наконец, дождался недолета. Набавлять начал опять по десятке и опять с одним и тем же исходом: недолет.

Потом за дисплей снова сел я. Через 13 попыток, когда очередное набранное мною число появилось на дисплее, под ним вспыхнуло: УГАДАЛИ.

Художник с изумлением смотрел на это слово, которого добивался от машины чуть ли не полчаса. Ему хотелось самому загадать и ввести в машину какое-нибудь число. «Хорошо, — сказал я, отвернувшись от дисплея. — Наберите на клавиатуре A =, потом свое число, потом двоеточие, потом вот такую фразу:

PRINT CHR ⊗ (22) : GOTO 20.

В конце нажмите клавишу ВК.

— Набрал, — оторвался от клавиатуры художник. — Скажите, а что это за светлый квадратик, который передвигался по экрану, когда я нажимал клавиши?

— Это *курсор*. Если нажать клавишу с каким-то символом, то он появится на экране в том месте, где в данный момент находится курсор. Одновременно курсор смещается на одну позицию вправо, указывая место, где появится следующий выводимый на дисплей символ.

Между тем на дисплее строчкой ниже загорелась надпись: «1-Я ПОПЫТКА?». Но мой гость смотрел не на нее, а на меня:

— PRINT — это ведь английское слово? «Печатать» — не правда ли? GOTO — «идти к», так?

— Совершенно верно. В основу языка Бейсик, которому обучены многие персональные компьютеры, и мой в том числе, положено несколько общеупотребительных английских слов. В Бейсике они называются *служебными словами* и используются в составе *операторов* — тех предписаний, которые человек сообщает машине. Занумерованные строчки из операторов образуют *программу*. Вы видели, как я вводил ее в машину: набирал строчку за строчкой, после набора каждой нажимал клавишу ВК, т. е. «возврат каретки». Потом, когда была нажата клавиша RUN, т. е. «запуск», а вслед за ней — ВК, машина принялась за выполнение содержащихся в программе пред-

писаний в порядке возрастания номеров строк. Если же набрать операторы без номера, как это только что сделали вы, — они выполнятся тотчас, стоит лишь нажать клавишу ВК.

Я повернулся к дисплею и принялся за отгадывание числа, которое загадал мой гость.

Через те же тринадцать попыток под введенным мною последним числом 1111 на дисплее светилось: УГАДАЛИ. «Вы загадали именно это число, 1111?» — для уверенности спросил я. «Да», — подтвердил он.

Тогда я встал из-за дисплея и протянул ему руку:

— Поздравляю! Вы уже начали программировать!

— Да?! Но я вроде бы еще ничего не делал!

Стараясь придать своему голосу побольше торжественности, я разъяснил:

— Когда вы набрали на клавиатуре A = 1111, вы ввели в машину *оператор присваивания*. Согласно ему переменной величине, стоящей слева от знака равенства, придается значение, указанное справа от знака равенства. Это значение задается либо конкретным числом, как у нас сейчас, либо *выражением*, составленным из нескольких переменных. Его надо вычислить при тех значениях, которые к данному моменту имеют эти переменные. Строго говоря, оператор присваивания должен начинаться со слова LET, что в переводе с английского означает «пусть». Но поскольку это самый распространенный оператор, было решено ради упрощения вводить его без начального слова. А машину научили поступать так: если она видит предписание без какого-либо служебного слова, то понимает его как оператор присваивания. В программе, по которой играет машина, загадываемое число обозначается именно буквой A. Когда вы нажали клавишу ВК, переменная A получила значение 1111 — так машина с вашей помощью и загадала то число, которое я потом отгадывал.

Я выдержал паузу, чтобы он лучше усвоил новую для него информацию. Потом продолжил:

— Когда вы набрали на клавиатуре GOTO 20, вы ввели в машину *оператор безусловного перехода* на указанную строку программы — 20-ю. Вы тем самым позволили компьютеру не выполнять предыдущую строку, избавили его от необходимости самому задумывать какое-то случайное число. Машина тотчас же совершила назначенный вами переход, когда вы нажали клавишу ВК.

Он был явно воодушевлен своим первым, пусть небольшим, успехом и захотел разобраться в программе игры.

— Нет, — сказал я ему, — сначала рассмотрим небольшую картинку, какую часто рисуют программисты. Она называется *блок-схемой*. Смотрите!

И я показал ему блок-схему игры «Угадай число» (рис. 1.3).

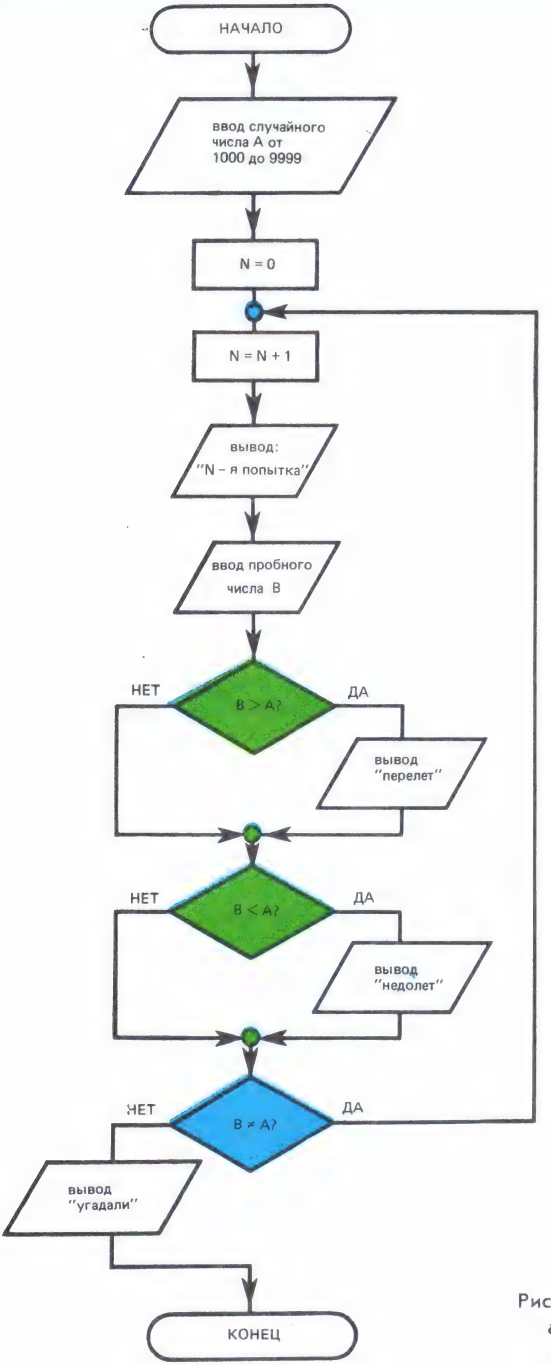


Рис. 1.3. Блок-схема алгоритма игры «Угадай число»

— Начинаем с «начала», — я указал на овал с такой надписью, потом на прямоугольник, нарисованный ниже. — Присваиваем переменной A произвольное значение с помощью функции RND(1). Я не хотел бы сейчас комментировать содержимое этого прямоугольника — оно пока что сложновато для вас. Следующий прямоугольник — новое присваивание, устанавливаем счетчик попыток на нуль: $N = 0$. Потом еще одно присваивание: заменяем N на $N + 1$. Обратите внимание: операторы присваивания пишутся в прямоугольных блоках. Результат первого выполнения оператора $N = N + 1$ равен единице. Следующий блок: выводим это число на дисплей, а за ним словосочетание «-я попытка?». Само по себе оно может показаться нелепым, но вместе с ранее выведенной цифрой выглядит как вполне осмысленный запрос: «1-я попытка?» Далее по схеме — ввод пробного числа B. Обратите внимание: в блок-схемах блоки ввода и вывода имеют форму параллелограмма. Ниже в нашей схеме вы видите несколько ромбов. В фигурах такой формы записываются условия. Из каждого ромба выходят две линии, помеченные словами «да» и «нет». Мы направляемся далее по той или другой линии в зависимости от соблюдения или несоблюдения условия. Итак, первый ромб — проверка: $B > A$? Если «да» — следите за надписями на соединительных линиях! — тогда на дисплей выводится сообщение «перелет», если «нет» — не выводится. Ромб второй: $B < A$? Если «да», на дисплее появится сообщение «недолет», если «нет» — не появится. Ромб третий: $B \neq A$? Если «да», значит, задуманное машиной число игроком не угадано, надо предоставить ему еще одну попытку. Проследите маршрут, намечаемый стрелкой «да»: счетчик попыток увеличивается на единицу, на дисплее появляется запрос, помеченный следующим номером, машина ждет новое пробное число. Если же проверка $B \neq A$? даст отрицательный результат, т. е. $B = A$, на дисплее загорится сообщение «угадали». Игра окончена, — я указал на овал с надписью «конец». — Вот и вся логика, по которой машина играет с человеком.

— Логика понятная. А как же программа?

— Программа получится, если мы ту же картинку перерисуем по-другому.

Я нарисовал прямоугольник, разграфил его и заполнил сектора строками программы (рис. 1.4).

— Это так называемая структурная диаграмма. Здесь нет никаких овалов, параллелограммов и ромбов. Все участки диаграммы имеют прямоугольные границы и плотно состыкованы друг с другом. Переходы, диктуемые соблюдением условий, совершаются по боковым дорожкам. Два верхних прямоугольника здесь такие же, как в блок-схеме, которую мы только что разбирали. В третьем прямоугольнике диаграммы объединились сразу несколько блоков прежней схемы, содержащих присваи-

```
10 A=1000+INT(9000*RND(1))
20 N=0
30 N=N+1:PRINT N"-Я ПОПЫТКА";:
   INPUT B
40 IF B>A THEN
          PRINT "ПЕРЕЛЕТ"
50 IF B<A THEN
          PRINT "НЕДОЛЕТ"
60 IF B<>A GOTO 30
70 PRINT "УГАДАЛИ"
```

Рис. 1.4. Программа «Угадай число»

вание N = N + 1, вывод на дисплей числа N и словосочетания «-я попытка». Отличие лишь в том, что в блоках прежней схемы мы описывали действия машины на русском языке, а здесь на Бейсике — языке, понятном компьютеру. Вы ведь с вашими знаниями английского уже разбираетесь в нем? PRINT в переводе с английского — «напечатать», с Бейсика — «вывести на дисплей». Это начальное слово *оператора вывода*. Далее — INPUT, начальное слово *оператора ввода*. Этого слова вы не найдете в словарях. Оно образовано от английского словосочетания «put in» — «помещать». Встретив его в программе, машина останавливается, выводит на дисплей текст в кавычках, стоящий после слова INPUT, и ждет, чтобы с клавиатуры была введена нужная информация. Число, которое вы вводили в ответ на очередной запрос машины, становилось значением переменной, указанной в конце оператора ввода. Не очень запутано?

Он отрицательно покачал головой, готовый слушать дальше.

— Тогда несколько слов о не очень заметных деталях. Вглядитесь в предпоследнюю строку, где обнаруживается несовпадение чисел A и B. Как видите, подобный факт в Бейсике выражается не привычным знаком ≠, а другим, образованным сочетанием знаков «меньше» и «больше», т. е. <>, а в некоторых версиях Бейсика ><. В символике этого языка вам встретится еще немало непривычных знаков. Далее: вы заметили, что третий прямоугольник уже первых двух? К нему справа тянется дорожка возврата от предпоследнего прямоугольника. Так на структурных диаграммах обозначаются переходы, которые на блок-схемах идут от ромбов наверх. В строке, от которой отходит такая дорожка, записывается какое-то условие. Если оно соблюдается — идем по восходящей дорожке к прямоугольнику, верхний край которого находится на одном уровне с ее концом. Если не соблюдается — переходим на нижележащий прямоугольник. Бывают в структурных диаграммах и нисходящие дорожки. Вот взгляните на четвертый прямоугольник нашей диаграммы: так изображаются переходы, которые в блок-схемах ведут из ромбов вниз. Проверяемое условие записывается в таких прямоугольниках вдоль верхнего

края. Видите: IF B > A THEN. В переводе на русский: «если B больше A, тогда». Это так называемый оператор *условного перехода*. Соблюдено условие, записанное после слова IF, — выполняются операторы, записанные в правом нижнем углу, не соблюдено — в левом. Он у нас пустой: при несоблюдении условия ничего делать не нужно.

— Все понятно, — резюмировал мой рассказ художник.

— А вот мне непонятно, почему вы не спрашиваете, зачем так много писанины? Блок-схема, структурная диаграмма, потом еще будет программа. . . Нельзя ли обойтись хотя бы без чего-то одного?

— Я думал, — удивленно посмотрел на меня художник, — если хочешь понять программирование, надо знать все — и блок-схемы, и структурные диаграммы. . .

— Отнюдь нет! Без блок-схем, например, можно обойтись вполне. Когда-то они были единственным средством, позволяющим наглядно представить логику программы. Им на смену пришли структурные диаграммы. Они не менее наглядны, а вдобавок удобны тем, что в них можно вписывать тексты программ. Потом достаточно пронумеровать прямоугольники, и содержащиеся в них записи станут строчками программы. В соответствии с этими номерами затем следует разметить *адреса переходов*. Вот, например, у нас в строке 60 стоит оператор GOTO — незавершенный, без указанного далее номера строки, на которую следует переходить. Дорожка от 60-й строки тянется к 30-й, так что дописываем: GOTO 30.

— Вы нумеруете строки как-то непривычно, — заметил художник.

— Да, числами, кратными десяти. Для машины в этом никакого неудобства нет: она будет выполнять строки программы в порядке возрастания их номеров, по какому бы закону эти номера ни возрастали. А вот для программиста, особенно неопытного, удобство немалое: если он при составлении программы пропустил какую-то строку, он может вставить ее между уже имеющимися, пометив промежуточным номером. Нам, к счастью, этого делать не придется: наша программа уже готова, и ее можно вводить в машину. Не попробуете ли? Только после каждой строчки не забывайте нажимать клавишу ВК. Ввели? Нажимайте RUN, потом еще раз ВК. Сыграйте, если хотите, снова.

И все повторилось точь в точь, как в тот раз, когда он впервые играл с компьютером. Он недоумевал, почему машина так долго изводит его сообщениями о недолете (видимо, число, задуманное ею, на сей раз оказалось очень большим), а дождавшись перелета, сердился, что и это сообщение долго не сходит с дисплея. Но сам его сердитый недоумевающий голос стал увереннее, и в нем сквозила радость: человек впер-

вые видел, как машина действует по программе, введенной им самим!

И снова после него за дисплей сел я. Теперь он следил за моими действиями придирчивым взглядом знатока.

— Когда я первый раз наблюдал вашу игру, — сказал он, — мне бросилось в глаза, с какой механической быстротой вы набирали свои пробные числа. Вы, наверное, владеете какими-то безошибочными правилами для поиска ответа. Почему вы всегда находите его именно за 13 попыток? Что это за мистическое число?

— Просто я знаю *алгоритм*, по которому загаданное число можно угадать за минимум попыток.

Вынув из ящика стола листок бумаги, я разделил его вертикальной чертой на две равные половинки:

— Вот, в сущности, и весь мой алгоритм. Всех четырехзначных чисел от 1000 до 9999 — девять тысяч. Свое первое пробное число я беру точно из середины этого ряда. По ответу машины, «недолет» или «перелет», заключаю, где находится задуманное число — среди первых или среди последних четырех с половиной тысяч. Отобранный числовой промежуток делю следующим пробным числом опять-таки пополам и тем самым сужаю интервал поиска. До единицы он сузится не более чем за 14 попыток, это можно рассчитать. Но скажите, зачем вам все эти расчеты и алгоритмы, если вы и так прекрасно научились общаться с машиной? Чего стоили бы компьютеры, если для работы на них требовались бы солидные познания? Тогда уж они наверняка не получили бы такого широкого распространения, которое мы видим сейчас!

— Я хочу сам составлять программы, — ответил он.

— Тогда нам придется встретиться с вами еще раз, и, пожалуй, не один.

Диалоговый режим

Человек задает машине вопрос. Машина, произведя необходимые выкладки, отвечает на него. Человек обдумывает полученный ответ и задает новый вопрос. Такой режим общения человека с машиной называется *диалоговым*. Он кажется естественным, чуть ли не единственно возможным. Между тем ЭВМ далеко не сразу приобрела способность к такому диалогу с человеком.

Можно ли по переписке сыграть с товарищем, живущим в другом городе, в игру «Угадай число»? Игра, конечно, будет скучная, но. . . почему бы и нет? Играют же по переписке в шахматы! Общение с ЭВМ первых поколений напоминало игру по переписке. Человек приходил в вычислительный центр, оставлял там две заранее подготовленные колоды перфокарт: одну — с пробитой на них программой, другую — с исходными данными расчета, а на следующий день (это в лучшем случае) получал от машины ответ в виде *распечатки*.

Чем не общение по переписке? Человек машину и не видел, она для него была как бы в другом городе. Такой режим работы на ЭВМ называется

пакетным. Оператор ЭВМ составлял из полученных от пользователей перфокарт единый пакет, машина считывала из него программу за программой. «Ни секунды простоя ЭВМ!» — такой лозунг витал в машинном зале, но его реализация давала плоды противоположного свойства. Не видя промежуточных результатов, пользователь не мог быстро выявлять ошибки, допущенные в программе или в исходных данных. А в случае ошибки полученный результат терял ценность, и расчет приходилось проводить заново. В результате значительную часть распечаток приходилось выбрасывать.

ЭВМ стали работать более продуктивно, когда их научили разделять свое рабочее время между многими пользователями, сидящими за выносными терминалами — устройствами ввода и вывода информации. Общение с машиной стало походить на телефонный разговор с иногородним абонентом через телефонистку, от которой нужно было дожидаться вызова.

Общение с персональным компьютером, диалоговый режим работы с ним подобны обычному телефонному разговору с товарищем, который всегда готов выслушать вас в любое удобное для вас время.

Бит

Если вести игру «Угадай число», деля пополам исходный и последующие числовые интервалы, то половинки могут оказаться неравными. Загадано, допустим, одно из трех чисел 1, 2, 3. Строго пополам их ряд не разделишь. Чтобы лучше понять выигрышную стратегию в этой игре, надо взять подходящий исходный интервал чисел. Рассмотрим пример, представленный на рис. 1.5. В ряду из восьми чисел загаданное отыскивается за три деления пополам. А теперь вглядитесь в такую таблицу:

$$8 = 2^3 \qquad \log_2 8 = 3$$
$$16 = 2^4 \qquad \log_2 16 = 4$$
$$32 = 2^5 \qquad \log_2 32 = 5$$

Становится ясным, как определить количество шагов, за которое выполняется подобный розыск в ряду четырехзначных чисел. Всего таких чисел 9000. Берем от этой величины логарифм по основанию 2. Получаем $\log_2 9000 = 13{,}135709$. Вот и выходит, что искомое четырехзначное число отыскивается за 13 — 14 вопросов, на которые можно дать лишь один из двух возможных ответов: «недолет» — «перелет», «да» — «нет» и т. п.

Рис. 1.5. Поиск загаданного числа методом половинного деления

Такие вопросы называются альтернативными. Минимальное число альтернативных вопросов, которое нужно задать для получения информации,

может служить мерой этой информации. Единица ее измерения называется бит.

Информация о том, где в ряду из восьми чисел находится какое-то одно определенное, составляет, таким образом, три бита. Информация о том, какое четырехзначное число задала машина, составляет 13,135709 бита.

Адрес ячейки

ЭВМ, ведя различные расчеты, играет сама с собой в игру «Угадай число» беспрестанно. Если, например, спросить у компьютера: «Машина, что ты хранишь в 109-й ячейке своей памяти?», то ЭВМ станет добираться до нужной ячейки как бы по сети дорожек, то и дело ветвящихся надвое (рис. 1.6). Любое число (и названный адрес тоже) машина представляет в двоичной системе счисления:

$$109 = 1101101$$

Как происходит перевод привычных для нас десятичных чисел в двоичную систему, мы поговорим позже. Здесь же отметим лишь то, что нуль в представлении адреса ячейки можно понимать как путь налево при ветвлении ведущих к ней дорожек, единицу — как путь направо.

В машине подобное ветвление реализовать несложно. Например, так: единица — заряженный конденсатор, нуль — разряженный. Или так: единица — намагниченное ферритовое кольцо, нуль — размагниченное. Или так: единица — триггер с напряжением на выходе, нуль — без напряжения.

Дойдя до нужной ячейки, ЭВМ узнает, что находится там, и сообщает об этом человеку.

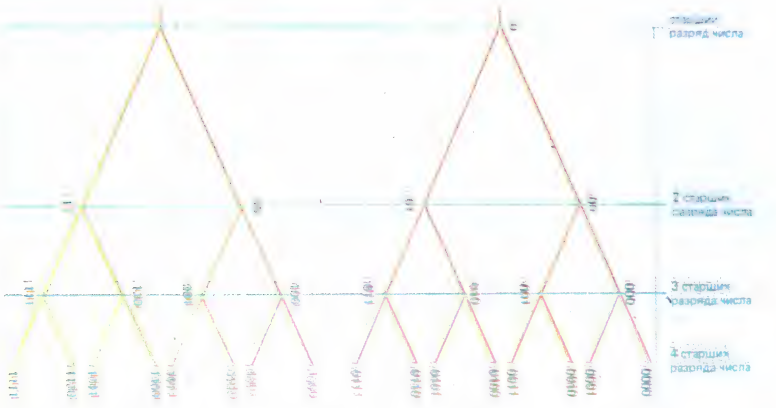


Рис. 1.6. Поиск ячейки памяти по ее адресу

Сколько развилок нужно пройти, добираясь до искомой ячейки, столько бит и составляет информация о ее адресе, столько разрядов содержит запись адреса в двоичной системе счисления. Общее же число таких адресов выражается степенью, в основании которой — двойка, а в показателе — число разрядов в записи адреса.

Этюд 2. САМАЯ ПОПУЛЯРНАЯ ПРОГРАММА

Когда я решил освоить Бейсик, мне посоветовали обратиться к одному программисту из нашего ВЦ, который, как все уверяли, сумеет научить меня этому языку за несколько дней.

— Вы кто по специальности? — спросил он меня.

— Инженер.

— С математикой, стало быть, знакомы? И квадратное уравнение знаете, как решать?

Я кивнул.

— Тогда с квадратного уравнения и начнем. Вот оно: $ax^2 + bx + c = 0$. Разберем программу для его решения.

```
10 PRINT "РЕШЕНИЕ КВАДРАТНОГО УРАВНЕНИЯ A*X*X+B*X+C=0"
20 INPUT "ВВЕДИТЕ ЧЕРЕЗ ЗАПЯТУЮ ЗНАЧЕНИЯ A,B,C",A,B,C
30 IF A=0 AND B=0 AND C=0 THEN 110
40 IF A=0 AND B=0 THEN 120
50 IF A=0 THEN 130
60 IF C=0 THEN 140
70 R=-B/(2*A):D=B*B-4*A*C:I=SQR(ABS(D))/(2*A)
80 IF D<0 THEN 150
90 IF D=0 THEN 160
100 PRINT "КОРНИ ДЕЙСТВИТЕЛЬНЫЕ X1=";R+I;"X2=";R-I:STOP
110 PRINT "КОРНИ МОГУТ ПРИНЯТЬ ЛЮБОЕ ЗНАЧЕНИЕ":STOP
120 PRINT "УРАВНЕНИЕ ВЫРОЖДЕНО":STOP
130 PRINT "КОРЕНЬ ЕДИНСТВЕННЫЙ X=";-C/B:STOP
140 PRINT "КОРНИ ДЕЙСТВИТЕЛЬНЫЕ X1=0 X2=";-B/A:STOP
150 PRINT "КОРНИ КОМПЛЕКСНЫЕ X1=";R;"+";ABS(I);"*I
    X2=";R;"-";ABS(I);"*I":STOP
160 PRINT "КОРНИ СОВПАДАЮТ X1=X2=";R:STOP
```

Рис. 1.7. Программа «Квадратное уравнение»

На листе бумаги было напечатано шестнадцать занумерованных строчек, состоящих из математических символов, английских и русских слов (рис. 1.7). Я удивился:

— Такое простенькое уравнение — и такая большая программа?

— А вы что же — хотели все уместить в одну строчку?

— Математикам это удается.

— Это только кажется.

И мой учитель написал на бумаге формулу для вычисления корней квадратного уравнения:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

— Знак «плюс — минус» — продолжал он, — как известно, означает, что первый корень вы должны вычислять по формуле со знаком «плюс», второй — по формуле со знаком «минус»:

$$x_1 = -\frac{b}{2a} + \frac{\sqrt{b^2 - 4ac}}{2a};$$

$$x_2 = -\frac{b}{2a} - \frac{\sqrt{b^2 - 4ac}}{2a}.$$

Так что одной строчкой тут никак не обойдешься.

— Ну, пусть будет две.

— И двух мало. Ведь чтобы складывать и вычитать половинки каждой формулы, — он подчеркнул выражения — $b/2a$ и $\sqrt{b^2 - 4ac}/2a$, — их надо сначала получить. Вот смотрите, — и он отчеркнул в программе (см. рис. 1.7) строку 70:

```
70 R = -B/(2*A) : D = B*B - 4*A*C : I = SQR (ABS (D)) / (2*A).
```

Я улыбнулся:

— Вы так и стараетесь написать подлиннее! Разве нельзя было в последнем равенстве поставить вместо D его выражение и приплюсовать к правой части то, что вы обозначили через R? Зачем вычислять отдельно эти D и R?

— Затем, что вы не можете знать заранее, какими будут коэффициенты вашего уравнения. Вдруг такими, что дискриминант $b^2 - 4ac$ окажется отрицательным? Тогда корни будут комплексными, и их придется вычислять совсем по другим формулам. Вот я и вычисляю дискриминант отдельно, чтобы потом проверить его знак. Если тот окажется отрицательным, я направлю вычислительный процесс по одному пути, если положительным — по другому.

Я снова углубился в строки программы.

— Вы заменили все строчные буквы прописными?

— В Бейсике строчные буквы для обозначения *переменных* обычно не употребляются. Да их и нет в печатающих устройствах большинства персональных компьютеров.

— А что это за звездочка?

— Знак умножения. Если его выражать традиционным крестиком, его легко будет спутать с буквой X.

— А почему нельзя было написать B^2 вместо B * B?

— В Бейсике все строчки должны быть одного уровня. Оттого и деление обозначается наклонной черточкой, а не горизонтальной, и скобки лишние появляются. А для степени значок есть. Можно написать B ↑ 2. Но произведение B * B машина вычислит в несколько раз быстрее.

— В строке 70 буква I или единица?

— Буква I, конечно.

— SQR — это квадратный корень?

— Да. Он извлекается из выражения, которое стоит вслед за ним в скобках.

— В скобках стоит ABS. Это что?

— Абсолютная величина.

— Она берется от величины, которая стоит дальше в скобках, от дискриминанта D. Но зачем брать от него абсолютную величину?

— На тот случай, если дискриминант окажется отрицательным. Давайте снова посмотрим на формулы для вычисления корней квадратного уравнения. Слева напишем их для случая, когда корни действительные, справа — когда комплексные. Видите: если дискриминант $D = b^2 - 4ac$ больше нуля, то корень извлекается из него, если меньше, то из величины −D. Вспомните определение абсолютной величины числа: она совпадает с ним, если число положительное, и отличается от него знаком минус, если отрицательное. Стало быть, какими бы ни оказались корни квадратного уравнения, действительными или комплексными, в формулах для них квадратный корень в обоих случаях извлекается из абсолютной величины дискриминанта. И всюду полученный корень делится на $2a$. Такую дробь, пригодную для обоих случаев, я и получаю в строке 70. Как видите, программисты тоже умеют быть экономными в своих текстах, как и математики.

— Вижу, убедился. Ну, а как машина поймет, что при положительном дискриминанте ей надо вести расчет по одним формулам, а при отрицательном — по другим?

— Читайте строку 80 и, если знаете английский язык, переводите.

— Если D < 0, то 150.

— Верно. Если выполняется условие, записанное после слова IF, то совершаются действия, описанные после слова THEN, а если после него стоит число, то происходит переход на строку с таким номером. Если же условие не выполняется, то все написанное после слова THEN игнорируется и происходит переход на следующую строку программы. Так устроен и функционирует *оператор условного перехода* IF . . . THEN.

Если за словом THEN стоит число, то его можно заменить на GOTO: IF . . . GOTO 150. Согласно нашей программе, если дискриминант D меньше нуля, следует перейти на строку 150, где запрограммированы соответствующие этому случаю формулы.

— А где предусмотрены формулы для случая, когда дискриминант положительный?

— Они запрограммированы в строчках, написанных после условного оператора.

Я прочел строку 90 и удивился вновь:

— Опять условный оператор!

— Да. Если дискриминант равен нулю, то корни квадратного уравнения совпадают, оба они равны величине R, вычисленной в строке 70. Этим случаем стоит заняться специально. В некоторых прикладных задачах совпадение корней уравнения имеет особый смысл.

— Если же дискриминант не равен нулю. . .

— И не отрицательный, как показала предыдущая проверка, то надо приступать к вычислениям по хорошо знакомым нам формулам. Сейчас мы разберем следующие две строки. Что вы ожидаете в них увидеть?

— Формулы для вычисления корней X_1 и X_2. Но ведь вы говорите, что в Бейсике все выражения должны быть одного уровня. Значит, подстрочных индексов писать нельзя?

— Это не самое страшное. Можно обозначить корни X1 и X2. Как бы вы записали следующие строки?

— Ну, допустим, так:

```
100 X1 = R + I
110 X2 = R - I
```

— Верно. Этими действиями машина получит оба корня. Но как их узнаете вы?

— Понимаю. Нужна команда, которая выведет полученные величины на дисплей.

— Выведет только их? Только два числа? Скажем, так: 2, 3. Но подумайте, если бы вы хотели кому-то сообщить решения квадратного уравнения, вы записали бы их иначе: $X_1 = 2$, $X_2 = 3$. Почему бы не научить машину, чтобы она выдавала результаты в такой форме? И давайте подумаем еще вот о чем. Когда вы сообщаете машине коэффициенты уравнения, которое собираетесь решать, вы еще не знаете, какими окажутся его корни — действительными или комплексными. Почему бы перед выводом корней на дисплей не отметить их характер?

И он показал на программу:

— Видите в строке 100 слово PRINT? Этот оператор выведет на дисплей, во-первых, все, что написано далее в кавычках; во-вторых, значение алгебраического выражения, указанное далее, значение суммы R + I, т. е. первый действи-

тельный корень квадратного уравнения. Допустим, первый корень оказался равным 2. На дисплее появится:

КОРНИ ДЕЙСТВИТЕЛЬНЫЕ X1 = 2

Я перебил его:

— А дальше. . . пусть второй корень равен 3. . . появится X2 = 3.

— Вы делаете успехи! После этого расчет по программе остановится — видите слово STOP?

— Теперь давайте посмотрим, что там у вас в программе ниже, как вычисляются комплексные корни, — предложил я.

— Но прежде, — уклонился он, — давайте посмотрим, что написано выше. А перед этим поговорим о том, как же возникло то «огромное» количество строк программы, которое вначале вас так напугало.

— Я уже понял, откуда они: надо ведь, чтобы программа годилась для любого сочетания коэффициентов уравнения. Для каждого сочетания нужно написать свой фрагмент программы.

— Точно. А еще раньше надо написать строки, в которых будет проверяться характер заданного машине сочетания коэффициентов.

— Строки вида IF. . . THEN?

— Они самые. А перед ними — строки с сообщениями, которые будут выведены на дисплей и подскажут вам, что нужно делать. Кстати, мой вам совет: когда станете сами составлять программы, не скупитесь на такие примечания и подсказки. Ну что же, теперь, надеюсь, вы без страха будете глядеть на полный текст программы.

И он придвинул его мне:

— Итак, строка 10. Фраза в кавычках выводится на дисплей. Она как бы приглашает вас к работе.

— Следующая строка начинается со слова INPUT.

— И тоже содержит фразу в кавычках. Она тоже будет выведена на дисплей. В ответ вы должны набрать на клавиатуре через запятую значения переменных, указанных в операторе INPUT. В нашем случае машина присвоит введенные значения переменным A, B, C и в следующих строчках начнет проверять, с каким же сочетанием коэффициентов она на сей раз имеет дело. В программе надо учесть все возможные сочетания, даже самые нелепые.

— Например, A = B = C = 0.

— Да, с этого и начинается анализ набора коэффициентов. Строка 30: IF A = 0 AND B = 0 AND C = 0 THEN 110, т. е. если A = 0 и B = 0 и C = 0, то перейти на строку 110. Прочтите ее, что появится на дисплее?

— КОРНИ МОГУТ ПРИНЯТЬ ЛЮБОЕ ЗНАЧЕНИЕ.

— Строка 40: IF A = 0 AND B = 0 THEN 120. Переходим

к строке 120 и читаем на дисплее: УРАВНЕНИЕ ВЫРОЖДЕНО.

— Если A = 0, — продолжил я, — то уравнение превращается в линейное, это и предусмотрено строкой 50: IF A = 0 THEN 130. . . Переходим на строку 130. . . Появится надпись: КОРЕНЬ ЕДИНСТВЕННЫЙ X =, а дальше — значение этого корня, дробь — C/B.

— Не составите ли сами фрагмент на случай C = 0?

Я отодвинул листок с программой.

— Попробую:

```
60 IF C = 0 THEN 140
140 PRINT «КОРНИ ДЕЙСТВИТЕЛЬНЫЕ X1 = 0 X2 = »;  — B/A : STOP
```

— И у меня так. Обратите внимание: если бы в моей программе эти строки были написаны неверно, то вам достаточно было бы набрать их заново под теми же номерами, и тогда при выполнении программы мои строки были бы заменены вашими. Это произошло бы благодаря следующей особенности Бейсика: строки написанной на нем программы можно набирать в любой последовательности — машина расставит их сама по порядку номеров, а если какие-то две строки были набраны с одним номером, то в программе под таким номером окажется та, что была введена последней. Ну вот, осталось разобрать фрагмент для случая отрицательного дискриминанта. Внимательно смотрите, как и что будет появляться на дисплее, не запутайтесь в кавычках! Пусть, например, R = 5, I = 2, т. е. корни равны $5 \pm 2i$.

— Сейчас попытаюсь.

И я медленно, то и дело заглядывая в программу, написал на бумаге: КОРНИ КОМПЛЕКСНЫЕ X1 = 5 + 2 * I X2 = 5 — —2 * I.

— А вы внимательны! — похвалил меня мой учитель. — Ну и совсем понятно теперь — посмотрите на строки 90 и 160, — что при D = 0 на дисплее появится: КОРНИ СОВПАДАЮТ X1 = X2 = . . . и далее значение корня, равное R.

— Посмотрим, как работает программа? — предложил я.

— Пожалуйста. Сейчас введу ее в машину.

Когда программа была введена и запущена, на дисплее мгновенно появилось:

```
РЕШЕНИЕ КВАДРАТНОГО УРАВНЕНИЯ A * X * X + B * X + C = 0
ВВЕДИТЕ ЧЕРЕЗ ЗАПЯТУЮ ЗНАЧЕНИЯ A, B, C?
```

Я набрал на клавиатуре 0, 0, 0. Тотчас на дисплее загорелось: КОРНИ МОГУТ ПРИНЯТЬ ЛЮБОЕ ЗНАЧЕНИЕ. На сочетание 0, 0, 1 машина столь же мгновенно отреагировала сообщением: УРАВНЕНИЕ ВЫРОЖДЕНО, в ответ на 0, 1, 2 вывела КОРЕНЬ ЕДИНСТВЕННЫЙ X = —2, на 1, 2, 0 выдала КОРНИ ДЕЙСТВИТЕЛЬНЫЕ X1 = 0 X2 = — 2.

Я набрал по памяти 1, — 2, 1 — коэффициенты квадрата

разности $(X - 1)^2 = X^2 - 2X + 1$, и получил в ответ: КОРНИ СОВПАДАЮТ X1 = X2 = 1. В той же триаде изменил второй коэффициент на — 3, и машина столь же быстро справилась с новым заданием:

X1 = 2.618034 X2 = .38196601

— Обратите внимание, — показал на дисплей мой наставник, — в десятичных дробях, начинающихся с нуля целых, нуль опускается, он не пишется в программе и не выводится на дисплей. Попробуйте теперь ввести числа 1, — 2, 5.

На дисплее возникло:

КОРНИ КОМПЛЕКСНЫЕ X1 = 1 + 2 * I X2 = 1 — 2 * I.

— Проверять будете? — улыбаясь, спросил меня хозяин машины.

— Все верно! — в тон ему ответил я.

Байт

Сколько информации вводим мы в память компьютера, когда нажимаем одну клавишу на его клавиатуре? Чтобы ответить на этот вопрос, поставим его сначала несколько иначе: сколько информации требуется, чтобы указать определенную клавишу? Сколько бит должна содержать эта информация?

Поставив искомое количество бит показателем степени двойки, мы должны получить в результате число, не меньшее, чем количество знаков, изображенных на клавиатуре.

Мы видим на ней русские и латинские буквы, цифры, знаки препинания, графические символы. Если пересчитать лишь буквы — русские и латинские, прописные и строчные — наберется 114 знаков. Добавим сюда цифры и знаки препинания — и получится больше, чем 128. Но 128 — это 2^7. Следовательно, семи бит информации нам явно не хватит. А восемь — вполне достаточно: $2^8 = 256$. В этом одна из причин, по которой программисты, ища единицу информации более крупную, чем бит, положили ее равной 8 битам. Назвали эту единицу байт.

Язык вычислительной машины — это язык двоичных чисел, цепочек нулей и единиц. Каждая клавиша на клавиатуре для машины имеет свое имя, свой двоичный код. В этом коде — восемь разрядов. Меньше, как мы видели, нельзя.

Когда мы нажимаем клавишу на клавиатуре компьютера, мы вводим в его память 8 бит, или 1 байт информации.

Символы Бейсика

Набор символов, употребляемых при записи программ на Бейсике, не так уж велик (см. 2-ю сторону обложки). Некоторые клетки помещенной там таблицы разделены вертикальной чертой — в них соседствуют варианты символов, принятые в различных диалектах Бейсика.

Читатель не обнаружит здесь ни символа квадратного корня, ни вертикальных скобок, выражающих абсолютную величину огражденного ими числа, ни квадратных скобок, обозначающих целую часть заключенной в них величины. Для употребляемых в Бейсике математических функций приняты однотипные выражения (см. 2-ю сторону обложки). Эти функции называются встроенными, поскольку алгоритмы их вычисления хранятся в компьютере, «встроены» в него.

Работающие с Бейсиком говорят также о *функциях пользователя*, или нестандартных функциях. Каждый волен определять их сам в программе, по которой ведется расчет. Для этого в программе пишется оператор описания нестандартной функции. Он начинается со слова DEF (сокращенное definition — определение). Далее следуют буквы FN (сокращенное function — функция) и за ними — еще одна буква, отличающая данную конкретную функцию. Затем в скобках, через запятую перечисляются ее аргументы, затем через знак равенства пишется в виде формулы алгоритм вычисления ее значений.

Например, если по ходу расчета вам требуется использовать отсутствующие среди встроенных функций вашей машины гиперболические синус и косинус, определяемые формулами

$$\text{sh } x = \frac{e^x - e^{-x}}{2}, \quad \text{ch } x = \frac{e^x + e^{-x}}{2},$$

то удобно ввести соответствующие *нестандартные функции* — назовем их FNS и FNC:

```
DEF FNS(X) = (EXP(X) - EXP(- X) )/2
DEF FNC(X) = (EXP(X) + EXP(- X) )/2
```

Если по ходу выполнения программы в ней встретится, допустим, выражение Z = FNS(Y), с величиной Y будут произведены все те действия, что и с аргументом X в определяющем эту функцию выражении, а результат будет присвоен переменной Z.

Знаки препинания в программах

Символика Бейсика включает знаки, заимствованные из различных областей знания: математики (плюс, минус, больше, меньше, равно), грамматики (точка, запятая, точка с запятой) и даже музыки (диез).

Есть тут и знаки, не вызывающие стойких ассоциаций с другими их применениями. С ними проще всего. Надо лишь раз и навсегда выучить, какой смысл придается им в Бейсике. Скажем, звездочка — это символ умножения, крышечка и стрелка — употребляемые в разных версиях языка символы возведения в степень.

Труднее всего со знаками, которые имеют хождение во многих дисциплинах, но в Бейсике понимаются не совсем так, как там. Вот, например, точка. В Бейсике ею при записи десятичных дробей отделяется целая часть числа от дробной.

Сложную роль играют в Бейсике запятая и точка с запятой, употребляемые в операторах ввода и вывода информации. Предположим, в программе есть строка

```
10 INPUT «ВВЕДИТЕ ДЛИНУ, ШИРИНУ, ВЫСОТУ», A, B, C
```

Мы уже знаем: встретив слово INPUT, компьютер прерывает выполнение программы и выводит на дисплей текст, написанный вслед за этим словом в кавычках. Но закончится ли на том вывод, зависит от того, какой знак препинания стоит в строке далее. Если запятая, то в конце выведенного текста появится знак вопроса, если точка с запятой — никакого дополнения не будет.

Запрошенные значения указанных величин следует вводить, перемежая запятыми. Если вы не хотите вводить новые значения всех переменных, перечисленных в операторе ввода, наберите вместо ненужного значения одну лишь запятую. Так, если в нашем примере вы наберете 10.5,,3.25 — переменная A примет значение 10.5, переменная C — значение 3.25, величина переменной B не изменится, останется равной, допустим, 1.61.

Пусть далее в программе есть фрагмент:

```
20  PRINT  «A =»;
30  PRINT  A,
40  PRINT  B
50  PRINT  C
```

Первым из этих операторов вывода (строка 20) на дисплей будет выдан текст, заключенный в кавычки. (Вообще где бы ни встретился в программе текст в кавычках, в операторе вывода или ввода, он переносится на дисплей без изменений, но при этом лишается кавычек.)

Затем согласно строке 30 выводится значение переменной A. Точка с запятой, которой завершен предыдущий оператор вывода, предписывает выводить новую информацию сразу вслед за старой, в той же строке.

Строка 30 в нашем примере завершается запятой. Этот знак велит помещать выводимую информацию в специальные зоны, на которые поделена строка дисплея. Таких зон пять, длина каждой — 8 или 16 позиций, в зависимости от того, сколько всего позиций в строке — 40 или 80. Каждая запятая — это указание начать размещение очередного значения с первой позиции следующей зоны. Если запятых несколько, то пропускается несколько зон.

Строка 40 не завершается никаким знаком препинания. Это сигнал к тому, чтобы продолжить вывод с новой строки.

В итоге (строка 50) на дисплее появится:

```
A = 10.5          1.61
3.25
```

Эти тонкости надо знать, тем более что их не так уж много. Было бы совсем хорошо, если бы дополнительные сложности не появлялись из-за обилия версий Бейсика. В некоторых из них вместо кавычек пишутся апострофы. В некоторых двоеточие выводится на дисплей вместо вопросительного знака при запросе информации, в других случаях оно применяется в роли математического символа деления, а операторы в строке разделяются косой чертой (см. рис. 1.8).

Впрочем, когда приступаешь к работе на конкретной машине, подобные особенности усваиваешь быстро.

Этюд 3. О РЫБАКАХ И РЫБКЕ

Переписка между читателем журнала и редактором

Уважаемый товарищ редактор!

В 10-м номере вверенного Вам журнала за 1984 год помещен кроссворд с фрагментами (с. 136 — 138). Позиция 28 по горизонтали гласит:

«ЗАДАЧА. Три рыбака легли спать, не разделив добычи. Проснувшийся первым решил взять свою долю и уйти. Но число рыб не делилось на три, и для этого он выбросил одну, а от остатка забрал треть. Второй и третий сделали то же самое. Найти наименьшее число рыб, допускающее такую операцию.

ОТВЕТ. Минус две рыбы. Выбрасывая одну, получаем минус три, забирая из них треть, оставляем минус две. Такое можно проделывать вновь и вновь (автор решения).»

Из ответов на кроссворд, опубликованных в следующем номере журнала, я узнал, что автор оригинального решения не кто иной, как выдающийся английский физик Поль Дирак, тот самый, кто предсказал античастицы. Напоминая читателям журнала любопытный факт его биографии, Вы хотели сказать: вот, мол, какое оригинальное мышление было у Дирака; дескать, именно способность к нетривиальным решениям позволила ему додуматься и до антирыб, и до античастиц.

Все это я понимаю. И тем не менее хочу заявить Вам: в опубликованном Вами решении задачи что-то не так. Или ответ неверный, хотя он оригинальный и требует нетривиального мышления, или формулировка задачи неточная. Судите сами.

Решение я проверял с помощью персонального компьютера, для чего составил программу на языке Бейсик. Условия задачи расширил: рыбаков может быть любое количество; каждый из них перед своим уходом по-английски может выбросить любое число рыб. M (число рыбаков) при условиях задачи из кроссворда равно трем, P1 (изменение в улове перед уходом очередного рыбака) равно минус единице.

Алгоритм решения задачи простой. Он проистекает из самого ее характера. А характер у нее явно циклический: каждый рыбак совершает те же действия, что и остальные. Значит, эти действия можно записать в программе только единожды и повторять столько раз, сколько нужно. В нашей задаче — M раз. Как известно, для этого перед цепочкой строк программы, описывающих повторяющиеся действия (программисты называют ее *телом цикла*), надо написать *заголовок цикла* FOR C = 1 TO M, а завершить эту цепочку следует строкой NEXT C. Два этих оператора руководят выполнением *цикла*. Вначале *параметр цикла* C равен единице (FOR C = 1), с каждым прохождением тела цикла он увеличивается на единицу (NEXT C), а когда он достигнет M (...TO M), произойдет *выход из цикла*, т. е. начнут выполняться операторы, записанные в программе далее, вслед за телом цикла.

В моей программе до входа в цикл задается начальное число рыб в улове P0, а в процессе выполнения цикла проверяется, удовлетворяет ли оно условиям честного дележа. Да — ответ готов. Нет — исходное число рыб P0 уменьшается на единицу, проверка повторяется.

Вот моя программа:

```
10 INPUT «M?»; M
20 INPUT «P1?»; P1
30 INPUT «P0?»; P0
40 P0 = P0 + 1
50 P0 = P0 — 1
60 P = P0
70 FOR C = 1 TO M
80 P = P + P1 : REM КОЛИЧЕСТВО РЫБ ИЗМЕНЕНО НА P1
90 P = P — P / M : REM РЫБАК ЗАБРАЛ СВОЮ ДОЛЮ
100 IF P <> INT(P) GOTO 50 : REM ДЕЛЕЖ НЕ ПОЛУЧАЕТСЯ
110 NEXT C
120 PRINT «ОТВЕТ = »P0
```

За честностью дележа в моей программе следит, если можно так выразиться, строка 100. Здесь проверяется, делится ли величина P на M после того, как она изменилась на величину P1. Для этого P, полученное в результате выполнения строки 90, сравнивается со своей целой частью INT(P). Если они не совпадают, значит P — ве́личина нецелая. А поскольку в строке 90 мы получаем ее, вычитая дробь P/M из целой поначалу величины P, то нецелой является и дробь P/M. Значит, P на M не делится, надо испытывать следующее P0, т. е. возвращаться на

строку 50, что и делается с помощью оператора безусловного перехода GOTO 50.

Обратите внимание еще на одну тонкость. Я учел, что перед первым же испытанием заданное число P0 уменьшается на единицу. Чтобы, несмотря на это, проверка началась именно с P0, я еще раньше (строка 40) увеличиваю его на единицу.

Все остальное в моей программе довольно очевидно, все вычисления просты. К тому же каждую строчку в теле цикла я снабдил *ремаркой* (REM).

Введите текст программы в свой компьютер и проверьте интересующие вас варианты. При условиях задачи, какими они заданы в кроссворде, расчет дает не одно, а целый ряд исходных количеств рыб:

. . . ,52,25,—2,—29,—56,—83,—110,—137, . . .

Здесь нет ни конца, ни начала, есть симметрия относительно минус двух.

Никто не будет спорить, что минус 29, а тем более минус 137, меньше, чем минус два.

Если б рыбаков было четверо (M = 4), то перед отходом ко сну им нужно было бы выловить . . . ,509,253, — 3, — 259, — 515, . . . рыб. У пяти рыбаков план улова был бы еще сложнее: . . . , 9371, 6246, 3121, — 4, — 3129, — 6254, — 9379, . . .

Итак, задача в формулировке кроссворда ответа не имеет. Чтобы ответ все-таки равнялся минус двум рыбам, условие задачи нужно уточнить, а именно: «Найти наименьшее по абсолютному значению число рыб». Но тогда оно будет содержать подсказку, намекающую на существование антирыб и сводящую тем самым на нет весь смысл задачи.

С уважением (подпись)

Уважаемый товарищ читатель!

Прежде всего благодарю Вас за критику. Со своей стороны, могу заметить, что и составленная Вами программа тоже не свободна от недостатков. Присмотритесь к строке 100. Здесь при несовпадении величин P и INT(P) происходит выход из цикла по оператору GOTO еще до того, как параметр цикла C достиг заданного максимального значения M. Но ведь подобный досрочный выход из цикла — это грубое нарушение принципов структурного программирования!

Вы сочли необходимым напомнить мне в своем письме, как работает пара операторов FOR C = 1 TO M и NEXT C. Позволю себе отметить, что Вы забыли упомянуть про полный вид первого из них: FOR C = 1 TO M STEP N. Здесь число N указывает, насколько должен измениться параметр C при каждом прохождении цикла. Если же добавка STEP отсутствует, то по умолчанию считается, что N = 1.

С уважением (подпись)

Уважаемый товарищ редактор!

Вы совершенно правы, отмечая допущенные мною ошибки и недомолвки. О программировании циклов я действительно сказал не все. Я опустил некоторые очевидные моменты, придерживаясь великого *принципа умолчания*, столь характерного для Бейсика.

Что же касается правил структурного программирования, то я, смею заверить, осведомлен о них не хуже Вашего, что и собираюсь доказать приведенным ниже новым вариантом своей программы (рис. 1.8). На сей раз за честностью дележа следит специально введенная мною переменная E (как посвященные, мы можем назвать ее своим именем: *признак*). Перед выполнением цикла она полагается равной нулю, *обнуляется*.

```
10 PRINT "ЧИСЛО РЫБАКОВ";\INPUT M
20 PRINT "НАЧАЛЬНОЕ ЧИСЛО РЫБ";\INPUT P0
30 PRINT "СКОЛЬКО РЫБ КАЖДЫЙ РЫБАК
   ВЫБРАСЫВАЕТ (С МИНУСОМ)";\INPUT P1
40 LET P0=P0+1\REM ШАГ ВПЕРЕД
50 LET P0=P0-1\REM ШАГ НАЗАД
60 LET E=0\REM ПРИЗНАК ДЕЛЕЖА ПО УСЛОВИЮ ЗАДАЧИ
70 LET P=P0\REM ПЕРВЫЙ РЫБАК НАСЧИТАЛ P0 РЫБ
80 FOR C=1 TO M\REM ПЕРЕБОР ВСЕХ РЫБАКОВ
90 LET P=P+P1\REM КАЖДЫЙ РЫБАК ВЫБРОСИЛ P1 РЫБ
100 LET P=P-P/M\REM ВЗЯЛ СВОЮ ДОЛЮ
110 IF P>INT(P) THEN E=1\REM ОСТАТОК ЦЕЛОЕ ЧИСЛО ?
120 NEXT C\REM ПРОСЫПАЕТСЯ СЛЕДУЮЩИЙ РЫБАК
130 IF E=1 THEN 50\REM ДЕЛЕЖ НЕ ПОЛУЧАЕТСЯ
140 PRINT "ОТВЕТ ";P0;" РЫБ"
```

Рис. 1.8. Программа «Рыбаки и рыбка»

В строке 110 если P не совпадает со своей целой частью, выполняется поставленный мною далее оператор присваивания E = 1. Цикл при этом не прерывается, а продолжает выполняться. Когда произойдет выход из цикла, то в строке 130 будет произведена проверка: чему же равен признак E? Единице? Значит, хотя бы один раз число P не было равно своей целой части. Следовательно, как я уже аргументировал в предыдущем письме, нецелой была дробь P/M и дележа не получится. Ну, а если переменная E сохранила нулевое значение, приданное ей до вхождения в цикл, значит, число P0 выдержало испытание. В последней строке программы оно выводится на дисплей.

В заключение довожу до Вашего сведения, что новая программа дает те же результаты, что и старая. Так что все мои претензии, высказанные в предыдущем письме, остаются в силе.

С уважением (подпись)

О структурном программировании и принципе умолчания

Что представляют собой правила *структурного программирования*, к которым апеллируют в своей переписке читатель и редактор журнала? И что это за великий *принцип умолчания*, на который ссылается читатель?

Важные правила и принципы программирования лучше усваиваются, когда накопишь определенный опыт составления программ, на деле убедишься в необходимости и пользе каких-то строгих канонов. Поэтому разговор о правилах структурного программирования будет начат на последующих страницах книги. Там же мы не раз поясним примерами принцип умолчания.

Ремарки в программах

Ремарки — это такие участки программы, которые компьютер игнорирует при ее выполнении, но выводит на дисплей в полном тексте программы (*листинге*). Ремарки в программе подобны по своей сути ремаркам в тексте пьесы — действующие лица их не произносят, но они помогают успешно поставить спектакль, лучше передать мысли автора. Можно так удачно расставить ремарки в программе, что для полного раскрытия ее особенностей не понадобится никаких дополнительных объяснений. Следует только помнить, что счет по программе с ремарками длится несколько дольше. С другой стороны, в программе, совсем лишенной ремарок, через некоторое время не разберется даже сам автор.

У ремарок есть еще одно полезное применение. Когда при отладке программы нужно узнать, что произойдет с расчетом, если изъять какой-то оператор, то его из программы можно не вымарывать — достаточно перед ним вставить слово REM, и машина проигнорирует его, сочтя за *комментарий*. Потом, если слово REM убрать, оператор вновь заработает.

Транслятор

Символ за символом набираем мы на клавиатуре компьютера программу, написанную на Бейсике. Но машина не понимает, что GOTO — это «идти к», а IF — это «если». Для нее привычен язык нулей и единиц.

Перевод с Бейсика на этот язык выполняется в компьютере с помощью специальной программы, называемой *транслятором*.

Сегодня известны десятки языков, на которых можно писать программы, — так называемых алгоритмических языков. Это и Бейсик, и Аналитик, и Паскаль, и Фортран, и Ада и др. Чтобы машина понимала тот или иной из них, нужно ввести в нее соответствующий транслятор — переводчик конструкций алгоритмического языка, понятных человеку, в цепочки нулей и единиц, понятных машине.

Этюд 4. КОМПЬЮТЕРНЫЙ РЕБУС

В школьные годы меня здорово злили головоломки, которые печатаются на последних страницах научно-популярных журналов. Я жадно читал такие журналы, от корки до корки, хотел знать все, прорабатывал каждую статью, а когда добирался до развлекательных разделов со всяческими головоломками, ребусами, шарадами, то недоумевал: зачем нужны пустые забавы? Уж лучше бы заняли это место еще одной статьей!

Отношение к подобным развлечениям у меня изменилось, когда в школе учился мой сын. Уму непостижимо, сколько задают на дом школьникам! И когда я видел сына, увлеченно решающего какой-нибудь пустяковый ребус, я думал: пусть хоть это отвлечет его от бесконечного поглощения знаний.

Но вот сын перешел в девятый класс. В расписании появился новый предмет: информатика. И мое отношение к головоломкам из научно-популярных журналов изменилось вновь. Я с удивлением обнаружил, что они не так уж бессодержательны! Разгадывание буквенно-цифровых ребусов представлялось мне не иначе, как перебор числовых комбинаций в арифметическом устройстве ЭВМ, а за раздумьями сыщика над обстоятельствами таинственного преступления виделись логические операции булевой алгебры.

Будь моя воля, я в изобилии иллюстрировал бы подобными головоломками скучные страницы нынешних учебников по информатике. На самых простых примерах можно наглядно показать весьма существенные особенности решения задач на ЭВМ, сотрудничества человека с машиной.

Чисто «машинное» решение задачи может в корне отличаться от «человеческого». У людей есть интеллект, смекалка, интуиция, наконец. Компьютер же может похвастаться пока лишь точностью и быстродействием. Оптимальное решение задачи достигается сочетанием ценных качеств машины и человека. Хорошие тому примеры доставляют, на мой взгляд, буквенно-цифровые ребусы. Мне вспоминается один из них, опубликованный в журнале «Квант»:

$$\begin{array}{r} \text{USA} \\ +\ \text{USSR} \\ \hline \text{PEACE} \end{array}$$

Нужно определить, какие цифры скрываются за латинскими буквами. Разным буквам, естественно, соответствуют разные цифры.

Конечно, эту задачу можно решить чисто «по-машинному» — перебором всех вариантов, последовательно придавая каждой из переменных A, C, E, P, R, S, U значения от нуля до девяти и отбирая те комбинации, которые отвечают условию.

Просто, но уж очень долго! Придется примерно десять миллионов раз делать такую проверку. А это пока многовато даже для самого современного персонального компьютера.

Оставим на время компьютер в покое и обратимся к логике. Без всякой ЭВМ ясно, что P может быть только единицей, а U — девяткой. Если это так, то E = 0. Проанализировав самый правый столбец букв в условии задачи, заключаем, что R = 10 — A.

Можно попытаться составить еще несколько соотношений между буквами и, решив их, найти окончательный ответ. Но не слишком ли долго продлятся такие поиски? Не привлечь ли для этого компьютер?

Варьируя значение A, C, R, S в пределах от 2 до 8 (ведь 0, 1 и 9 уже закреплены за E, P и U соответственно), машина станет проверять соотношение $900 + 10S + A + 9000 + 100S + 10S + R = 10000 + 100A + 10C$. Если оно выполняется, компьютер заменит буквы цифрами в равенстве USA + USSR = PEACE и выведет его на печать. Если соотношение не выполняется, машина приступит к проверке следующего сочетания букв. Все это можно записать одной строчкой программы:

```
IF 900 + 10 * S + A + 9000 + 100 * S + 10 * S + R =
    10000 + 100 * A + 10 * C THEN
PRINT U S A «+» U S S R «=» P E A C E
```

Эта строчка и составит тело цикла, по которому лучше всего вести поиск ответа.

Полный перебор всех вариантов потребует $7 \cdot 7 \cdot 7 = 343$ проверки. Для человека это по-прежнему многовато, а для машины — ничуть. По крайней мере можно поручиться, что львиная доля времени, затраченная на решение задачи, уйдет на

составление программы и ее ввод в память ЭВМ и лишь малая толика — собственно на расчет.

Варьируемых переменных у нас не одна, а целых три: A, C, S (R не в счет, поскольку она выражается через A соотношением R = 10 — A). Чтобы проварьировать все три, надо заключить тело цикла в тройную рамку из операторов FOR. . . TO и NEXT, по одной паре операторов для каждой переменной. Строго говоря, это будет уже три цикла, вложенных друг в друга, как матрешки. А еще раньше надо написать строчку, где величинам E, P, U будут присвоены значения 0, 1 и 9 соответственно:

```
10 P = 1 : E = 0 : U = 9
20 FOR A = 2 TO 8
30 R = 10 — A : FOR S = 2 TO 8
40 FOR C = 2 TO 8
50 IF 900 + 10 * S + A + 9000 + 100 * S + 10 * S + R =
10000 + 100 * A + 10 * C THEN PRINT  U S A «+» U S S R «=» P E A C E
60 NEXT C
70 NEXT S
80 NEXT A
```

Вводим эту программу в машину, нажимаем клавишу RUN, и за несколько секунд на дисплее появляются четыре варианта ответа:

932 + 9338 = 10270
966 + 9664 = 10630
977 + 9773 = 10750
988 + 9882 = 10870

Верен лишь первый вариант. В остальных разным буквам соответствуют одинаковые цифры, а это не согласуется с условием задачи.

Как отсеять лишние решения!

Откуда взялись лишние, неверные решения буквенно-числового ребуса? В составленной для его разгадки программе значения переменных не проверялись на совпадение. Исправим этот недочет: если, например, очередное C совпало с A, S или R, пропустим такой вариант и тут же перейдем к строчке, где увеличивается значение C:

IF C = A OR C = R OR C = S THEN 60.

Работая по исправленной программе (рис. 1.9), ЭВМ выдаст единственное решение головоломки — верное.

В дальнейшем вы не раз будете находить в тексте книги задачи для самостоятельного решения. Умейте только их искать! А для этого читайте книгу повнимательнее, повдумчивее.

«Снисходительность» Бейсика

Заметили ли вы, уважаемый читатель, что в операторе печати, записанном в строке 50 (см. рис. 1.9), переменные не отделяются друг от друга никакими знаками препинания? На их месте стоят пробелы, вместо которых более аккуратный программист должен был бы поставить точки с запятой.

Почему возможно такое вольное общение с ЭВМ? Благодаря принципу

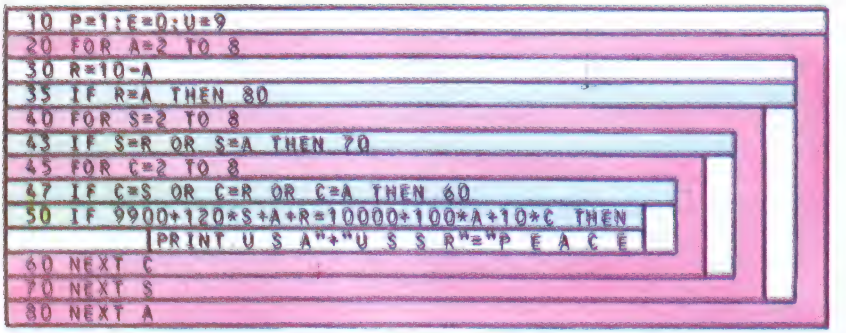


Рис. 1.9. Программа «Головоломка»

умолчания, столь характерному для Бейсика. Если человек забыл дать машине какую-то инструкцию, то она может сама решить, как ей в такой ситуации поступать, «вспоминая» типичные прецеденты.

Из Бейсика от версии к версии исключаются служебные слова, на месте которых не может стоять ничего другого: LET, THEN и т. д. В последних версиях за словом NEXT можно не указывать переменной — машина сама знает, что там должно стоять.

Выполнение оператора печати:

PRINT «Я ЗНАЮ БЕЙСИК

не приведет к аварийному останову, так как машина поймет, что человек просто забыл в конце фразы поставить кавычки: кроме них там ничего другого стоять не может.

Этюд 5. «НАЧАЛЬНИК» ПРОГРАММЫ

Я заведую лабораторией в большом НИИ. Если я буду вникать во все мелкие детали той работы, которую выполняет каждый из моих подчиненных, у меня на это просто не хватит времени. Я должен лишь знать «входы» и «выходы» каждого из них — извините за радиотехнический жаргон, т. е. должен знать, какое задание можно дать тому или другому сотруднику и какой результат можно от него ожидать. Когда лаборатория берется за новую задачу, я должен продумать стратегию ее решения. Разбить ее на подзадачи, наметить участки работы. Расставить по ним своих подчиненных сообразно с возможностями каждого. Создать каждому сотруднику наилучшие условия деятельности. И главное — суметь организовать всех в единый коллектив, обеспечить их четкое взаимодействие.

Вот с такой позиции могу посоветовать каждому, кто программирует на Бейсике: чувствуйте себя «начальником» той программы, которую составляете. Очень часто ее приходится собирать из *модулей* — блоков, которые уже кто-то сделал до вас. Прежде всего выясните, что должно подаваться на вход каждого блока, т. е. с какими исходными данными он работает, и что у него на выходе, т. е. какие результаты он выдает. Иногда можно и не знать во всех подробностях, как блок устроен. Но если отдельные блоки не сопрягаются друг с другом, приходится в них разбираться и кое-что изменять. Собрав все блоки в одну программу, надо обязательно обдумать, в каких условиях придется ей работать и нельзя ли чем-то улучшить ее функционирование.

Пример? Пожалуйста. Рассмотрим задачу о разложении целого положительного числа на простые множители.

Разрабатываем стратегию решения задачи, т. е. составляем алгоритм. Пусть он будет таким: брать одно за другим простые числа, начиная с двойки; взяв очередное, пробовать, делится ли на него без остатка исходное число. Если не делится, испытывать следующее простое число. Если делится, выписывать этот делитель, а с частным работать далее как с исходным числом. Если в результате очередного деления получится единица, прекратить процесс.

Внимательно, фраза за фразой читаем описание алгоритма. «Брать одно за другим простые числа». Кто у нас занимается простыми числами? Один из моих подчиненных недавно составил программу для проверки числа на простоту:

```
10 INPUT «ВВЕДИТЕ ПРОВЕРЯЕМОЕ ЧИСЛО N?» ; N
20 FOR J = 2 TO SQR (N)
30 IF N / J = INT (N / J) GOTO 60
40 NEXT J
50 PRINT N« — простое число» : GOTO 10
60 PRINT N« — составное число» : GOTO 10
```

Это не совсем то, что нужно. На вход подается произвольное число N, а на выходе появляется суждение о его простоте. Нам же требуется, чтобы на выходе возникали простые числа одно за другим. И тем не менее программу можно использовать — надо лишь несколько ее модифицировать. Будем подавать на вход целые числа одно за другим, начиная с двойки, и проверять их на простоту. Если проверка удалась, передаем это число на выход. Не удалась — испытываем следующее N:

```
10 N = 1
20 N = N + 1
30 FOR J = 2 TO SQR (N)
40 IF N / J = INT (N / J) GOTO 20
50 NEXT J
60 . . .
```

Выход у этого блока — строка 60, где начнется описание тех операций, в которых будет использовано новое N.

Нумерацию строк пока сохраним. Перенумеровать еще успеем. При этом, конечно, надо тщательно следить за адресами переходов, чтобы они изменились в строгом соответствии с общей перенумеровкой.

Проверим: верно ли работает такой блок, все ли простые числа он выдает? Нет, не все. В их ряду не будет двойки: при N = 2 и J = 2 условный оператор в строке 40 передаст управление на строку 20, а не на выход. Исправляем этот недочет:

```
10 N = 1
20 N = N + 1 : IF N = 2 GOTO 60
30 FOR J = 2 TO SQR (N)
40 IF N / J = INT (N / J) GOTO 20
50 NEXT J
60 . . .
```

Теперь все верно. Можно, конечно, усовершенствовать этот блок, ускорить его работу, проверять вслед за N = 3 лишь нечетные N. Но если машина достаточно быстродействующая, можно оставить все как есть.

Возьмемся за разработку следующего блока. Снова вчитываемся в описание алгоритма: взяв очередное простое число (у нас оно обозначено буквой N), пробовать, разделится ли на него без остатка исходное число (обозначим его A). Если не делится, испытывать следующее простое число. Если делится, выписывать этот делитель (т. е. выводить на дисплей, поскольку задача решается на компьютере), а с частным работать так же, как с исходным числом.

В этом «так же» сразу видится присваивание A = A / N ... Но нет, давайте все обговорим по порядку.

Условие делимости A на N без остатка выражается равенством A / N = INT (A / N). С него и начнем первую строку нового блока, а перед равенством напишем слово IF. Все, что нужно сделать при выполнении этого условия, запишем в той же строке далее после слова THEN — вывести N на дисплей, т. е. PRINT N, заменить A на частное от его деления на N, т. е. A = A / N.

То, что нужно сделать при невыполнении условия, запишем в следующей строке. Там, очевидно, должен стоять оператор безусловного перехода туда, где берется новое N, т. е. где к прежнему N прибавляется единица, — на строку 20 предыдущего блока.

Все, по-моему, изложено настолько подробно, что написать разрабатываемый блок не составляет труда:

```
60 IF A / N = INT(A / N) THEN PRINT N; : A = A / N
70 GOTO 20
```

Следующий блок: если в результате очередного деления A на N получается единица, прекратить процесс. Для этого блока, пожалуй, никаких новых строк не понадобится. Просто припишем в конце строки 60 предыдущего блока еще один условный оператор:

```
60 IF A / N = INT (A / N) THEN PRINT N; : A = A / N :
IF A = 1 THEN STOP
```

Тогда при A, равном единице, выполнение программы закончится, а на дисплей будет выведен полный список простых сомножителей, на которые разлагается исходное число.

Таким образом, мы наметили концовку программы. Начать ее нужно, конечно, со строки, в которой вводится число A. Напишем ее под номером, с которого обычно начинают программы на Бейсике, — под десятым:

```
10 INPUT«A?» ; A
```

Строка под таким номером уже была в блоке выдачи простых чисел. При сборке всех блоков в единую программу составляющие их строки придется перенумеровать, исправляя при этом все

адреса переходов, записанных после слов GOTO. Это сделать несложно. Сложнее другое.

Наша лаборатория — лишь одно из доброй полсотни подразделений большого НИИ. Если директор института будет вникать во все детали работы всех институтских лабораторий, отделов, служб, у него на это не хватит времени. Не зная до мелочей всех наших обстоятельств, он может отдать распоряжение, которое, мягко говоря, не лучшим образом отразится на нашей работе. Я как начальник должен быть начеку и, если нужно, добиваться, чтобы директор скорректировал свое неудачное распоряжение.

Тот, кто занимается программированием, большинство программ составляет по чьему-то распоряжению, для других людей. Если вы чувствуете себя «начальником» составляемой вами программы, вы должны отдавать себе отчет в том, что ее заказчики и пользователи могут разбираться в программировании хуже вас, могут и вовсе быть дилетантами в этом деле. Вы должны внести в свою программу нечто такое, чтобы она отказывалась выполнять заведомо неразумные распоряжения своего пользователя, ведущие к неверным результатам, сбоям и «авостам». Это называется у программистов *«защита от дурака»*, или, если сказать помягче, «защита от неверных действий человека».

Взять, к примеру, все ту же программу, которую мы составляем для разложения целых положительных чисел на простые множители. Она *зациклится*, если исходное число A нецелое или неположительное. Не имеет смысла обращаться к ней и в том случае, если A равно единице. Защиту от таких бессмысленных исходных данных можно устроить с помощью одной строки:

```
20 IF A<=1 OR A<>INT (A) THEN STOP
```

Теперь все готово к сборке программы. План этой работы прост: после только что написанной нами проверочной строки остается написать составленные нами блоки, подчиняя их строки и адреса переходов новой, единой нумерации. Кстати, на строку 20 разумно переходить после присваивания A = A / N для проверки, не оказалось ли A равным единице. Если это так, произойдет останов. В результате получается программа, представленная на рис. 1.10.

```
10 INPUT "A?";A
20 IF A<=1 OR A<>INT(A) THEN STOP
30 N=1
40 N=N+1:IF N=2 THEN 80
50 FOR J=2 TO SQR(N)
60 IF N/J=INT(N/J) GOTO 40
70 NEXT J
80 IF A/N=INT(A/N) THEN PRINT N;:A=A/N:GOTO 20
90 GOTO 40
```

Рис. 1.10. Программа «Простые множители»

Проверим ее работу на числе 1237600. *Протокол* расчета выглядит так:

```
RUN
A? 1237600
2 2 2 2 2 5 5 7 13 17
```

Как упростить программу!

Программисты говорят, что всякую программу можно сократить хотя бы на один оператор. (Всегда ли следует это делать, вопрос особый.) Не хотите ли вы, читатель, потренироваться в этом искусстве? Присмотритесь к только что разбиравшейся программе разложения числа на простые множители. Даже если вы не сумеете добиться ее сокращения на один оператор, вы вполне сможете сократить ее на несколько символов.

В самом деле, в строке 20 вместо A <> INT (A) можно написать A > INT (A). Знак < тут лишний, поскольку никакое число не может быть меньше своей целой части. В той же строке вместо A <= 1, если уж число A целое, можно написать A < 2.

Внимательно читайте приводимые в книге программы. Возможно, вам удастся в чем-то их усовершенствовать. Это будет свидетельствовать о том, что программирование вы усваиваете неплохо.

Жаргон программистов

Кому-то режет слух выражение «защита от дурака». Кого-то удивляют термины «обнулить», «аварийный останов» (или, как коротко выражаются программисты, «авост»). Кто-то впервые слышит слово «зациклиться».

Приглашаем таких читателей заглянуть в конец книги, где помещен краткий толковый словарь встречающихся в ней профессиональных терминов и жаргонных словечек, употребляемых программистами.

Этюд 6. ТИШЕ ЕДЕШЬ — ДАЛЬШЕ БУДЕШЬ

Не скрою: люблю пофилософствовать. Это оттого, считают сотрудники нашего отдела, что я хорошо освоил свою персональную ЭВМ, очень быстро справляюсь с ее помощью со всеми заданиями и у меня остается много свободного времени.

Философствовать я собираюсь как раз на тему о том, как экономить время, работая на ЭВМ.

Продолжительность счета на машине — это еще не все время, которое уходит на решение задачи. Ведь если задача новая, то программу взять бывает неоткуда. Ее приходится составлять самому. Длительность работы над ней тоже следует учитывать, когда подсчитываешь время, потраченное на задачу. Многие об этом забывают и ставят перед собой единственную цель: написать такую программу, которая работала бы как можно быстрее. Сразу это, как правило, не удается. Начинаются поиски новых, более быстродействующих алгоритмов и способов их программной реализации.

И вот тут могут случиться любопытные казусы. Вы улучшаете программу до такой степени, что она выдает результат за считанные секунды. Но на оптимизацию и отладку вы затратили несколько часов. Первый же, несовершенный вариант требовал для счета нескольких минут, и его составлением вы занимались тоже несколько минут.

Так какой же, спрашивается, вариант лучше — совершенный или несовершенный?

Один мой приятель говорит в таких случаях: совершенствовать программу таким путем — это все равно, что потратить

полчаса на выбор оптимального сочетания средств транспорта, чтобы попасть с одной улицы на другую, когда пешком этот путь можно проделать за десять минут.

Разумеется, я здесь не говорю о программах, предназначенных для многократного использования: сокращая их сроки выполнения даже на секунды, мы в итоге экономим многие часы. Но если программа будет работать только раз — зачем тратить час ради экономии одной минуты?

И уж совсем не дело, когда за верх совершенства принимаются алгоритмы, выдуманные в «домашинную» эру, рассчитанные на ручное выполнение. Человек работает по ним довольно быстро и сноровисто, а перевести их на язык ЭВМ бывает не легко, и на это уходит много времени.

Чтобы ввести программу в машину, тоже требуется время. Причем тем больше, чем длиннее программа. Если при вводе допущена ошибка, какое-то время уходит на ее нахождение и исправление. Вероятность случайной ошибки пропорциональна длине программы. Поэтому программы стараются делать покороче. Но и тут надо знать меру. Если работа по сокращению программы затягивается надолго, то она становится уже бесцельной.

В общем если учесть, что время решения новой задачи на ЭВМ складывается из времени написания и ввода программы и времени счета, то нередко оказывается, что простые, пусть довольно медленные и длинные программы позволяют решить задачу за кратчайшее время. Тише едешь — дальше будешь. Здесь вполне подходит эта популярная русская пословица.

Кроме того, простота решения дает субъективное ощущение его надежности.

Вы знаете, как определяется наибольший общий делитель двух целых положительных чисел? В «домашинную» эру было предложено несколько алгоритмов для этой цели. Можно, например, разложить каждое число на простые множители и отобрать в этих двух числовых наборах совпадающие числа. Их произведение даст искомый наибольший общий делитель.

Попробуйте составить программу для ЭВМ по такому алгоритму. Не получается? Тогда давайте действовать по-другому.

Возьмем меньшее из двух исходных чисел в качестве первого кандидата на звание наибольшего общего делителя. Проверим, делится ли без остатка на это число оба исходных. Если нет, заменим кандидата числом, на единицу меньшим, и повторим проверку на делимость. Рано или поздно она удастся.

Программа, по которой можно провести такой поиск (рис. 1.11), занимает всего шесть строк. Даже не очень опытный программист напишет ее за пять минут. По такой программе моя машина для чисел 1245 и 4220 нашла наибольший общий делитель 5 за 15 секунд.

```
10 INPUT "A,B?";A,B:HOD=B+1
20 IF A<B THEN
        HOD=A+1
30 HOD=HOD-1
40 IF A/HOD>INT(A/HOD) GOTO 30
50 IF B/HOD>INT(B/HOD) GOTO 30
60 PRINT "HOD=";HOD
```

Рис. 1.11. Программа «Наибольший общий делитель 1»

Было бы все-таки интересно составить программы по тому алгоритму, о котором я рассказывал вначале. Засеките время, которое уйдет у вас на ее составление. Уверен, что в пять минут вы не уложитесь!

Имя переменной

В тексте программы, по которой отыскивается наибольший общий делитель двух чисел (см. рис. 1.11), читателя может удивить обозначение искомой величины: HOD.

Непривычно, правда? До сих пор в любой из приводившихся программ именем любой переменной (его еще называют *идентификатором*) служила либо латинская буква, либо латинская буква с цифрой: A, P2, M0 и т. п. Это было правилом в самых первых версиях Бейсика. Это стало традицией в последующее время.

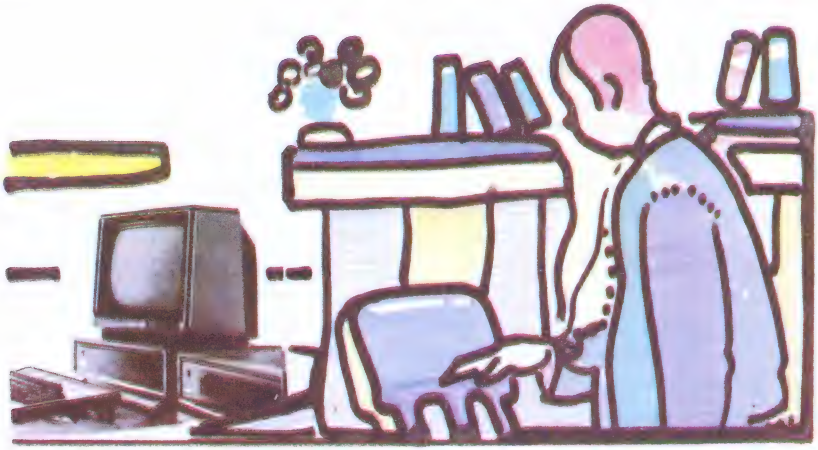
В последних версиях Бейсика идентификатором может быть цепочка букв и цифр длиной до 40 знаков: ALFA1, TEMPERATURA2 и др.

Правда, некоторые машины по-прежнему различают переменные только по первым двум знакам. Так что HOD и HOK остаются для них одной и той же переменной. Но тем не менее писать программы стало удобнее — видно, что содержит переменная. Когда читаешь программу поиска наибольшего общего делителя, то и без комментариев ясно, что HOD — это искомый делитель.

Цикл с параметром

Несколько слов о *цикле с параметром*, в полной форме записываемом с помощью двух операторов вида FOR I = 1 TO M STEP N. . . NEXT I.

Выход из такого цикла происходит, когда проверка параметра I показывает, что с очередным приращением N он переступил через установленное для него предельное значение M, говоря иначе, что $I + N > M$ или $I + N < M$, если N — отрицательно. При этом в разных версиях Бейсика по-разному решается вопрос о значении параметра после выхода из цикла. На одних машинах он становится равным $M + N$, на других такого прибавления не происходит. Поэтому при реализации алгоритмов не стоит ссылаться на значение параметра цикла после выхода из него.



Этюд 7. ЭВМ БРОСАЕТ ЖРЕБИЙ

Когда я писал книгу о программировании на Бейсике, то старался для каждого понятия из вычислительного дела подыскать аналогию из обыденной жизни. Оказалось, что это совсем не просто! Вот, скажем, цикл, многократное повторение одних и тех же операций. Тема в машинной математике очень важная, о ней необходимо поговорить обстоятельно, со множеством примеров. А откуда их взять? Даже в нудной, рутинной работе, когда одно и то же действие приходится повторять по многу раз подряд, так сказать, по одной и той же программе, человек не терпит однообразия. А характерный признак хорошо отлаженной машины, напротив, в том и состоит, что она повторяет свои действия с неукоснительным постоянством и производимые ею вещи невозможно отличить друг от друга. Однообразие — стихия машины, даже такой высокоразвитой, как электронная вычислительная. Сфере ее «размышлений» должны принадлежать продукты машинного производства — большие серии однотипных объектов, различающихся лишь номерами в техническом паспорте.

Представим себе, что завод выпустил тысячу станков. Их нужно распределить между заказчиками согласно имеющимся заявкам. Распределение поручено компьютеру. Как ввести в него информацию об этих станках? Обозначить каждый своей буквой? На это не хватит всех алфавитов мира! Да и требуется ли тут разнообразие символов? Станки-то все одинаковые! Поэтому их

лучше обозначить одной и той же буквой с различными числовыми индексами.

Конечно, мне могут возразить: абсолютно одинаковых предметов в реальном мире не бывает. Согласен. Но ведь вычислительная машина имеет дело не с реальной действительностью, а с ее математическими моделями, которые строит человек. А человеку удобнее работать с моделями попроще. Очень часто бывает вполне допустимо отождествить между собой объекты одной совокупности (как, допустим, станки одной серии), назвать их в программе одним именем, обозначить одним символом и различать по номерам, приписываемым к этому символу в скобках. Например, X(1), X(2), X(3) и т. д. Такие наборы в программировании называют *массивами*. Слагающие их переменные именуются *индексными* — в отличие от тех, что массив не образуют и потому называются *простыми*.

В программах на Бейсике переменные того и другого типа различаются без труда: за именем индексной переменной стоят скобки с указанным в них индексом, за именем простой переменной скобок нет.

В мысленном разговоре с воображаемыми оппонентами я приводил все новые примеры, представлял все новые ситуации, в математическом описании которых очень кстати было бы понятие массива. Я помнил и о своем намерении вводить каждое новое понятие через жизненную аналогию. Где в обыденной жизни мы встречаемся с большими совокупностями совершенно одинаковых занумерованных предметов? Ответ подсказало воспоминание о детстве.

. . . Маленький, меньше грецкого ореха, деревянный бочонок с цветными ободками и с номером, выпукло проступающим на обоих донцах. Россыпь таких фишек, издали неотличимых друг от друга, лежит на дальнем конце большого стола. Отец ссыпает их в мешочек, запускает туда руку и минуту-другую перемешивает его содержимое.

Когда я был ребенком, любимой игрой в нашей семье было лото. По вечерам все усаживались за столом, ведущий доставал из мешка одну за другой фишки с номерами, и комната оглашалась возгласами: «Восемь! Кочерга! Сорок один! Уточки! Тридцать два! Дюжина! Стульчики!». Перед каждым игроком лежала разграфленная карточка со вписанными в клетки числами от 1 до 90, по пять чисел на каждый из трех рядов в карточке. Называемые числа накрывались. И вот кто-то из игроков первым восклицал: «Квартира!». Это значило, что в его карточке накрыты уже четыре числа в каком-то ряду. А вскоре у кого-то оказывались закрытыми все пять чисел какого-то ряда. Такой игрок считался победителем. Снова фишки ссыпались в мешок, и начиналась новая игра.

Игра в лото — вот что мне позволит доходчиво рассказать о массиве. А примером того, как компьютер обращается с мас-

сивами, может стать программа, моделирующая действия ведущего. Пусть после ее запуска на дисплее возникают целые случайные числа от 1 до 90. Игрокам остается поглядывать на дисплей и накрывать демонстрируемые на нем числа на своих карточках.

На первый взгляд такая программа может содержать одно-единственное присваивание:

```
10 B = INT (1 + 90 * RND (1))
```

Функция RND (1) при каждом обращении к ней принимает случайное значение в интервале от нуля до единицы. Ее аргумент — формальный параметр. Если он равен нулю, то функция RND (сокращенное *random* — случайный) при каждом новом выполнении программы выдает один и тот же ряд чисел. При ненулевом — пропускает несколько первых чисел этого ряда; с какого места они начнут выдаваться, определяется, как правило, моментом запуска программы. В играх желательна неповторяемость ситуаций — оттого и на сей раз употреблен ненулевой аргумент: RND (1).

(Есть версии Бейсика, где функция RND используется без указания аргумента. В таких случаях для установки функции RND в произвольное место ряда случайных чисел используют оператор RANDOMIZE).

Следует отметить, что выдаваемые любой машиной последовательности случайных чисел всегда периодичны. И хотя повторяющийся фрагмент может быть весьма длинным (миллионы, миллиарды чисел), абсолютно случайными числами их не назовешь. Их именуют *псевдослучайными*, а порождающую их функцию — *датчиком* (*генератором*) псевдослучайных чисел. У нас эти числа лежат в интервале от нуля до единицы. После умножения на 90 и сложения с 1 из них получаются вещественные числа в интервале от 1 до 91. Функция INT округляет свой аргумент, т. е. берет от него целую часть. Так получается то, что нужно для игры: случайные целые числа от 1 до 90. Иными словами, строка 10 служит датчиком таких чисел, обозначаемых буквой B. Переменную B остается вывести на дисплей оператором PRINT и повторить ту же операцию вновь и вновь, совершая переход GOTO 10.

Программа вышла короткой и простой. Никакого массива в ней не понадобилось. Одно плохо: выдаваемые нашим датчиком числа могут повторяться. А в реальной игре это невозможно: в мешке для игры в лото нет фишек с одинаковыми номерами.

Как исключить повторения? Вот тут и пригодится массив. Пусть в нем будет столько же элементов, сколько фишек в мешке. Все элементы занумерованы и в начале игры равны нулю. Но как только программа выведет на дисплей очередной номер, то пусть элемент массива, помеченный данным номером, станет равным единице — для этого введем в нашу программу такой

оператор. Далее поставим оператор, который после выдачи очередного случайного целого числа B проверит, не равен ли единице элемент с таким номером. Не равен — значит, такого числа еще не было, выводим его на дисплей, равен — передаем управление на датчик случайных чисел, чтобы он выдал новое число.

В программе, где используется массив, его, как правило, сначала нужно ввести, т. е. присвоить ему имя и отвести место для его хранения в памяти компьютера. Это делается с помощью оператора DIM (от английского *dimension* — размер). Вслед за ним надо указать имя массива, а в скобках — число его элементов. Увидев такой оператор в программе, машина, во-первых, резервирует участок своей памяти для вводимого массива. Во-вторых, она автоматически присваивает его элементам начальные нулевые значения, а это как раз то, что нам нужно.

Вводим массив A(B) в первой же строке программы (рис. 2.1). Дальнейшие ее строки однозначно диктуются тем, что о ней было сказано выше.

```
10 DIM A(90):REM ВСЕ ФИШКИ В МЕШКЕ
20 FOR I=1 TO 90:REM ЗАГОЛОВОК ЦИКЛА
30 B=INT(1+90*RND(1)):
   IF A(B) GOTO 30:REM ЭТО УЖЕ БЫЛО
40 PRINT I;"-Я ФИШКА -";:A(B)=1
50 IF B=7 THEN PRINT "КОЧЕРГА":GOTO 160
60 IF B=11 THEN PRINT "БАРАБАННЫЕ ПАЛОЧКИ":GOTO 160
70 IF B=12 THEN PRINT "ДЮЖИНА":GOTO 160
80 IF B=13 THEN PRINT "ЧЕРТОВА ДЮЖИНА":GOTO 160
90 IF B=22 THEN PRINT "УТОЧКИ":GOTO 160
100 IF B=44 THEN PRINT "СТУЛЬЧИКИ":GOTO 160
110 IF B=69 THEN PRINT "ТУДА И ОБРАТНО":GOTO 160
120 IF B=77 THEN PRINT "ТОПОРИКИ":GOTO 160
130 IF B=80 THEN PRINT "БАБКА":GOTO 160
140 IF B=90 THEN PRINT "ДЕД":GOTO 160
150 PRINT B:REM ФИШКА БЕЗ НАЗВАНИЯ
160 FOR J=1 TO 1000:NEXT J:REM ЗАДЕРЖКА
170 NEXT I:REM КОНЕЦ ЦИКЛА
180 PRINT "МЕШОК ПУСТОЙ."
```

Рис. 2.1. Программа «Лото»: B — номер фишки; I — номер хода; J — счетный индекс паузы; A(B) — признак, вышла ли B-я фишка

Восемнадцать строк. . . Не много ли? Отнюдь нет, потому что в них учтены многие особенности игры, о которых говорится далее, в комментариях к строчкам программы.

Строка 20. Вынимаемым фишкам ведется счет. Ведется он в цикле со счетным индексом I.

Строка 30. Функция RND(1) в силу ненулевого значения своего аргумента в каждой новой игре выдает новый ряд чисел.

Это соответствует реальной игровой ситуации, где к началу новой игры фишки лежат в мешке не в том порядке, что к началу предыдущей. «Помешать фишки в мешке» можно, запустив, а через произвольный промежуток времени прервав выполнение короткой программы:

```
10 B = RND (1) : GOTO 10
```

Строка 30. В конце ее устроена проверка: не встречалось ли раньше число B, выданное предыдущим оператором? Иными словами, чему равен элемент A(B)? Если он отличен от нуля, запрашивается новое число:

```
IF A(B) = 1 GOTO 30
```

Строка 40. Сюда переходим, если A(B) по-прежнему равно нулю, т. е. число B не встречалось ранее. На дисплей выводится порядковый номер фишки I, а соответствующему элементу A(B) присваивается единичное значение.

Строки 50 — 140. Записанные здесь условные операторы узнают те фишки, которые носят специфические названия, и выводят их на дисплей вместо номера.

Строка 150. Остальные фишки демонстрируются на дисплее своими номерами.

Строка 160. «Пустой» цикл для задержки вывода очередного числа, чтобы игроки успевали считать с дисплея очередное появившееся на нем число и просмотреть свои карточки.

Строка 170. К счетному индексу I прибавляется единица и, если он не достиг заданного максимального значения, равного 90, совершается переход к заголовку цикла, на строку 20.

Строка 180. Сюда попадаем при I = 90. Все фишки вынуты из мешка, игру приходится прекращать. Такое, впрочем, случается очень редко. Обычно выигрыш наступает, когда мешок еще не пуст.

Массивы в разных диалектах Бейсика

Программы на Бейсике, написанные для разных машин, как правило, различаются какими-то деталями, порой довольно существенными. Поэтому принято говорить о разных версиях (или диалектах) Бейсика. Разнообразие его диалектов проявляется и в особенностях использования массивов. Перечислим некоторые из них:

1. Не в каждой версии Бейсика после задания массива все его элементы автоматически приравниваются нулю. Исключение составляет, например, машина ДВК. Такого рода редкие, но все же встречающиеся исключения заставляют опытных программистов гарантировать себя от неожиданностей и при необходимости «обнулять» только что заданные массивы, например

```
DIM A(23) : FOR I = 0 TO 23 : A (I) = 0 : NEXT I.
```

2. Минимальный индекс элемента обычно равен нулю, а на некоторых ЭВМ, например на «Искре 226», — единице, но есть версии Бейсика, где значение минимального индекса можно задать специально.

3. Максимально допустимый индекс массива может быть ограничен либо каким-то числом (например, в «Электронике ДЗ-28» — числом 255), либо объемом незанятого участка памяти машины, который можно целиком отвести под массив. Во многих машинах, оснащенных Бейсиком, объем этого остатка выражается в байтах специальной переменной, обозначаемой особым сочетанием символов — SIZE, FRE(0) и пр. Если возникает необходимость задать массив с максимально возможным значением индекса, то достаточно выполнить оператор

```
DIM A(SIZE/N)
```

Вместо N надо поставить конкретное число, указывающее, сколько байт займет в памяти отдельный элемент массива.

4. Не во всех версиях Бейсика можно задавать размер массива с клавиатуры, с помощью оператора INPUT. Например, так:

```
INPUT N : DIM A(N)
```

На «Искре 226» это невыполнимо, там размер массива следует указывать конкретным числом: DIM A(100), DIM Z(300) и т. п.

Системные переменные

Не очень часто, но всегда очень кстати применяются в Бейсике переменные, имеющие строго определенный смысл. Они обозначаются специальными сочетаниями символов. Мы только что познакомились с одной из таких переменных SIZE (размер) — она указывает программисту, какой объем оперативной памяти остался незанятым его программой и используемыми в ней переменными. Существуют и другие переменные такого рода: одна отсчитывает текущее время, другая при аварийном останове указывает, какой оператор послужил причиной сбоя, и т. д. Такие переменные называются системными.

Досрочный выход из цикла

Можно ли досрочно выходить из цикла? Правила структурного программирования диктуют однозначный ответ на этот вопрос: нет, нельзя. Алгоритм с досрочно прерванным циклом структурной диаграммой не изобразишь.

Иной недоверчивый читатель, возможно, спросит: а нужно ли столь неукоснительно придерживаться правил структурного программирования? Если в Бейсике допускаются переходы с помощью оператора GOTO на строки с какими угодно номерами, то почему же нельзя им пользоваться? В других алгоритмических языках, например Фортране, Алголе, тоже допустимы переходы по меткам (это понятие обобщает понятие номера строки).

Три основные причины заставляют программистов структурировать алгоритмы.

Во-первых, какие угодно переходы в программе приводят к тому, что через некоторое время в ней не сможет разобраться даже сам ее автор. Что же говорить о ком-то другом, кто захочет эту программу улучшить, что-то в ней изменить?

Во-вторых, руководствуясь принципами структурного программирования, несложно из готовых, уже кем-то разработанных и опробованных программ составлять новые, более сложные.

В-третьих, наиболее совершенные языки программирования, такие, как Паскаль или Ада, допускают переходы к меткам лишь в строго определенных, очень немногочисленных случаях.

Многие полагают, что без меток можно совсем обойтись. Считается, что квалификация программиста обратно пропорциональна числу использованных меток в его программе на Паскале, Аде или ПЛ / 1.

Кстати сказать, в Бейсике нет абсолютной свободы в использовании оператора GOTO. Ни на одной машине невозможно войти в тело цикла минуя его заголовок — возникнет аварийная ситуация, которую программисты называют «NEXT без FOR». (Вас удивляет, как можно смешивать в одном выражении слова разных языков? Программистов, к сожалению, это не удивляет).

Выходить из цикла, минуя его конец (NEXT), можно только в последних версиях Бейсика. Если предпринять такой выход на какой-нибудь старой машине, то при последующем входе в цикл возникнет другая аварийная ситуация: «Два раза FOR без NEXT». Особо опасно выходить досрочно из вложенного цикла. Это может создать трудно улаживаемую путаницу.

Опытный программист, работая даже с самыми совершенными версиями Бейсика, все равно не позволит себе досрочно прервать цикл с параметром. Ведь его программа может быть введена в машину, оснащенную не столь совершенной версией, и тогда возникнет сбой.

Этюд 8. ТРОЙНОЙ ВЫИГРЫШ

Не стану утверждать, что мои знания в программировании блестящи. Но метод, которым я самостоятельно осваивал эту науку, готов порекомендовать всякому. Метод простой. Узнав какое-то новое понятие, я всегда стремился с его помощью усовершенствовать хотя бы одну программу, которую составил прежде.

Выигрыш тут часто двойной, а то и тройной. Во-первых, я получаю прекрасное практическое упражнение на новую, только что освоенную тему. Во-вторых, пополняю свою библиотеку программ, обогащаю свои знания и навыки в программировании. А в-третьих, при модернизации программы за счет нового понятия, ее часто приходится дополнять деталью, в которой не нуждался прежний, простой вариант. За этой деталью нередко встает новое понятие, т. е. напрашивается новый шаг на том же пути. И изучение программирования неплохо идет вперед.

Когда я добрался до массивов, мне пришла в голову мысль: вот есть у меня программа, по которой отыскивается наибольший общий делитель двух чисел; напишу-ка я программу для определения наибольшего общего делителя сразу нескольких чисел.

Буду рассматривать их как некоторый числовой массив A(1), A(2), . . . , A(N). Первым кандидатом на роль наибольшего общего делителя назначу наименьшее из них. Обозначу его AMIN, т. е. А минимальное. Для наибольшего общего делителя сохраню обозначение HOD, как в прежней программе.

Проверку делимости чисел A (I) на HOD стану вести в цикле. Чтобы не нарушать правил структурного программирования, введу вспомогательную переменную — признак P. Перед входом в цикл положу P равным нулю. При невыполнении условия делимости стану присваивать этому признаку значение 1. После выхода из цикла предусмотрю проверку: равно ли P единице? Если P = 1, значит, не все элементы A(I) делятся на только что использованное число HOD, надо брать кандидатом на эту роль число, на единицу меньшее. Если P = 0, значит, все числа A (I) разделились без остатка на проверявшееся HOD, и его можно выводить на дисплей.

```
10 HOD = AMIN + 1
20 HOD = HOD - 1 : P = 0
30 FOR I = 1 TO N
40 IF A (I) / HOD > INT (A (I) / HOD) THEN P = 1
50 NEXT I
60 IF P = 1 GOTO 20
70 PRINT «HOD =»; HOD
```

Этот фрагмент я написал без труда, ведь он во многом похож на программу для определения наибольшего общего делителя двух чисел. Я набрал на клавиатуре этот фрагмент, внимательно прочитал весь текст, появившийся на дисплее, выискивая ошибки.

Надо признаться, ошибок при наборе я делаю много: то букву не ту наберу, то знак операции. Если таких опечаток немного, устранить их легко: подводишь курсор к неверно набранному знаку, нажимаешь соответствующую клавишу, и ошибочный знак заменяется нужным. Бывает и так, что заменять приходится целую строку. Это тоже несложно: набираешь ее заново, с тем же номером. Так как две строки не могут иметь один и тот же номер, то прежняя будет «вытолкнута» из памяти компьютера и замещена новой. Подобным же образом можно вычеркнуть из памяти лишнюю строку, введя вместо нее «пустую», т. е. состоящую только из номера. Так в памяти компьютера окажется нужная строка. Но на дисплее будут соседствовать обе — и старая, оставшаяся на прежнем месте в набранном прежде тексте программы, и новая, как бы «приписанная» к этому тексту снизу. Иногда у меня просто в глазах рябит от правленных-переправленных строк. В таких случаях я отдаю компьютеру команду LIST и потом нажимаю клавишу ВК. На дисплее под всеми строчками, появлявшимися на нем до этого, выстраивается отредактированный текст программы.

Итак, на дисплее передо мной светился именно тот фрагмент, который я составил для поиска наибольшего общего делителя. И хотя ошибок в нем уже не было, я вглядывался в него не без сомнений. Перед выполнением этого фрагмента, размышлял я, уже должно быть известно число AMIN, минимальное из чисел A (1), A (2), . . . , A (N). Как же его искать?

Подходящий метод вспомнился быстро. Вся его суть — в попарных сравнениях. Чтобы понять его, представим себе педанта, которому понадобилось найти в корзине с яблоками самое спелое. Он берет первое попавшееся, откладывает его и считает спелее всех. Потом вынимает второе яблоко и сравнивает с первым. Если новое окажется спелее, то им заменяется отложенное сначала, и та же процедура продолжается со следующими яблоками. Перебрав все, можно быть уверенным, что яблоко, отложенное последним, — самое спелое.

Запрограммировать этот способ было нетрудно. Сначала я оформил его в виде короткой программы и занес ее в свою библиотеку — на случай, если придется специально заниматься поиском наименьшего числа из числовой совокупности. Разработал также вариант для поиска максимума. А перед тем, как вставлять эту программу фрагментом в программу для поиска наибольшего общего делителя, еще и усовершенствовал его — объединил с вводом чисел A(I). Ввод постарался запрограммировать с максимальной заботой об удобствах пользователя: на дисплее появляется номер очередного вводимого числа, и в ответ нужно набрать это число на клавиатуре, а потом нажать клавишу ВК.

```
10 FOR I = 1 TO N
20 PRINT I« — Е ЧИСЛО» ; : INPUT A (I)
30 IF I = 1 THEN AMIN = A (1)
40 IF A (I) < AMIN THEN AMIN = A (I)
50 NEXT I
```

Я написал эти пять строк на бумаге, по привычке нумеруя их последовательными числами, кратными десяти. Тотчас вводить их с клавиатуры я не мог: они вытеснили бы строки, набранные прежде под теми же номерами.

Было ясно: введенный ранее текст надо передвинуть с занимаемого им места в памяти, чтобы нумерация строк начиналась, допустим, со ста. Для выполнения таких перемещений в Бейсике есть команда RENUM, вслед за которой указываются все необходимые для перенумерации данные. Символ за символом я набрал

```
RENUM 10, 100, 10
```

В результате выполнения этой команды будут перенумерованы уже находящиеся в памяти строки с номерами от 10 и далее, новая нумерация начнется с номера 100 и пойдет с шагом 10.

Я нажал клавишу ВК и получил новый листинг программы:

```
100 HOD = AMIN + 1
110 HOD = HOD — 1 : P = 0
120 FOR I = 1 TO N
130 IF A (I) / HOD > INT (A (I) / HOD) THEN P = 1
140 NEXT I
150 IF P = 1 GOTO 110
160 PRINT «HOD = »; HOD
```

Перечел строки, пронумерованные заново. Команда RENUM работает по-умному: меняет не только номера строк, но и адреса переходов. В конце строки 150 стоит GOTO 110 вместо прежнего GOTO 20: переход, как и требует смысл алгоритма, совершается к оператору, уменьшающему величину HOD на единицу.

Далее я набрал одну за другой строки фрагмента, предназначенного для поиска минимального из вводимых чисел. Эти строки, начальные в будущей единой программе, на дисплее опять-таки возникли вслед за набранным прежде текстом — завершающим фрагментом единой программы. Тогда я снова отдал компьютеру команду LIST, и те же строки повторились на дисплее ниже, выстроившись в порядке возрастания номеров.

И только тут я заметил свое упущение. В самом начале программы должен располагаться фрагмент, где вводится количество чисел, для которых отыскивается наибольший общий делитель, и задается их массив. Снова сдвигать строки? Нет, можно обойтись и без этого. Для записи обеих операций хватит одной строки. Если написать ее под номером 5, она займет должное начальное место:

```
5 INPUT «N?»; N : DIM A(N)
```

Теперь, кажется, все. Программа составлена. Только нумерация строк в ней не очень стройная. Отдать еще раз команду перенумеровки? Да, пожалуй, это стоит сделать.

Я набрал одно лишь слово RENUM без единого числа за ним: в таком случае все строки программы нумеруются заново последовательными числами, кратными десяти, как того требует традиция Бейсика (несколько устаревшая — она идет от тех времен, когда еще не было команды перенумеровки строк).

Выведя на печать окончательный вариант, я дополнил распечатку линиями, разбивающими текст программы на структурные блоки, и раскрасил их, чтобы был виден алгоритм задачи (рис. 2.2).

```
10 INPUT "N ?";N:DIM A(N)
20 FOR I=1 TO N
30 PRINT I"-Е ЧИСЛО";:INPUT A(I)
40 IF I=1 THEN
        AMIN=A(1)
50 IF A(I)<AMIN THEN
              AMIN=A(I)
60 NEXT I
70 HOD=AMIN+1
80 HOD=HOD-1:P=0
90 FOR I=1 TO N
100 IF A(I)/HOD>INT(A(I)/HOD) THEN
                        P=1
110 NEXT I
120 IF P=1 GOTO 80
130 PRINT "НОД="HOD
```

Рис. 2.2. Программа «Наибольший общий делитель 2»

Редактирование программы

Суждение автора представленной на рис. 2.2 программы о том, как лучше изучать Бейсик, заслуживает внимания. А вот сообщаемые им сведения о способах редактирования программ требуют обстоятельного комментария. Ведь в этом случае речь идет не о взглядах, которые можно разделять или не разделять, а о стандартных, общеупотребительных приемах.

На Бейсике из операторов можно составить программу, которая будет выполняться автоматически, а можно набирать их на клавиатуре, не предваряя номером строки, и выполнять тотчас в «непосредственном» режиме.

В таком режиме можно работать с компьютером, как с калькулятором, произведя оперативные расчеты. Например, для единичного расчета по формуле $c = \sqrt{a^2 + в^2}$, где $a = 5$, $в = 12$, можно выполнить такую последовательность команд:

```
A = 5
B = 12
C = SQR (A * A + B * B)
PRINT C
```

После набора каждого оператора следует нажимать клавишу ВК. После того как она будет нажата в последний раз, мы увидим на дисплее результат: 13.

В этом же «непосредственном» режиме проводится и редактирование программ с помощью команд LIST (в переводе — список, перечень; она выводит на дисплей программные тексты), RENUM (сокращенное renumber — перенумеровать; она изменяет нумерацию строк; в некоторых версиях Бейсика она называется RESEQ), DEL (сокращенное delete — вычеркивать; она стирает сразу несколько строк).

Каждая из этих команд может быть снабжена уточняющими примечаниями. К слову LIST можно приписать одно из таких выражений:

L1 (вывод одной строки с номером L1);

L1 — L2 (вывод строк с номерами от L1 до L2);

L1 — (вывод всех строк, начиная с L1);

— L2 (вывод всех строк от начальной до L2).

Точно так же задаются пределы стираемого фрагмента при использовании оператора DEL.

По ходу редактирования программы для поиска наибольшего общего делителя нескольких чисел дважды употреблялась команда RENUM. Сначала она была дана в полной форме:

RENUM L1, L2, M

Здесь L1 — номер первой из строк, подлежащих перенумеровке; L2 — номер, который получит первая из нумеруемых заново строк; M — шаг, с которым пойдет новая нумерация.

Второй раз слово RENUM было набрано без всяких примечаний. В этом случае перенумеровке подвергаются все строки программы с начала до конца, начальная строка получает номер 10 и приращение номеров равно 10.

Команду LIST также можно отдавать без примечаний. Тогда на дисплее, как мы уже знаем, появится полный текст программы.

Заметим, что команду RENUM удобно применять в тех случаях, когда возникает необходимость раздвинуть две строки с последовательными номерами, чтобы вставить между ними новую.

Математическое обеспечение ЭВМ

Искусство работы на ЭВМ состоит не только в умении писать программы, искать и исправлять в них ошибки. Гораздо важнее умение грамотно подбирать и использовать уже кем-то составленные и отлаженные программы.

Программа, по которой была однажды решена какая-то типовая задача, после ее использования не теряется, а заносится в *библиотеку стандартных программ*. Владелец компьютера постоянно расширяет такую библиотеку, заимствуя все новые программы у коллег, работающих на других ЭВМ. Но расширяет не бессистемно. За каждой конкретной задачей опытный программист видит целый класс родственных задач. Предвидя, что в будущем ему может встретиться любая задача из этого класса, он пополняет свою библиотеку сразу целым пакетом соответствующих прикладных программ.

Пакет прикладных программ подобен хорошо подобранной серии книг определенной тематики, ни одна из которых не дублирует другую, а все вместе они дополняют друг друга через ссылки, расставленные в разных местах текста. Любая программа из пакета может использоваться самостоятельно, а может быть подпрограммой решения более сложной задачи.

Весь комплекс программ, их описаний и инструкций к ним, обеспечивающих решение конкретных задач на данной ЭВМ, называется ее *математическим обеспечением*. Сюда, разумеется, входят и программы, которые хранятся в постоянном запоминающем устройстве машины, «вшиты» в нее, как говорят программисты.

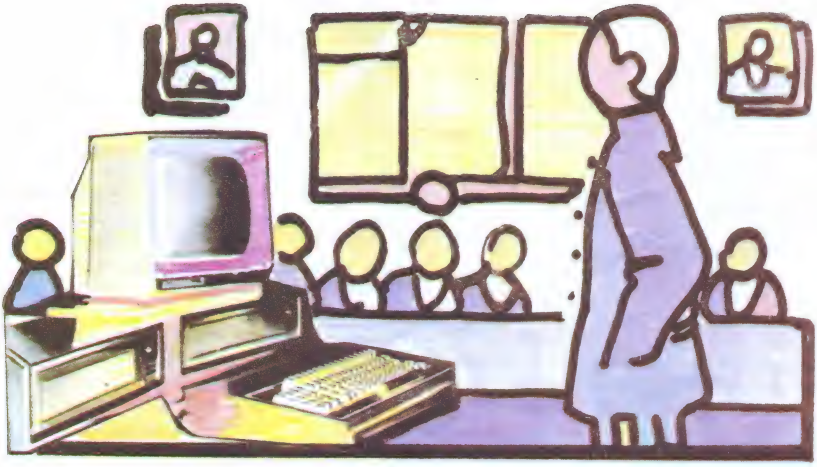
«Не красна ЭВМ объемом своей памяти и быстродействием, а красна своим математическим обеспечением», — так можно утверждать, перефразируя известную поговорку. ЭВМ без математического обеспечения подобна мастеру, лишившемуся навыков своего ремесла.

Почему не работает программа?

Предложим читателю небольшой экзамен.

Первый вопрос — теоретический. Почему программа «Наибольший общий делитель 2» будет работать не на всех машинах, «владеющих» Бейсиком? Если вы затрудняетесь с ответом, внимательно прочтите самую первую строку программы, а потом — комментарий на стр. 48.

Второй вопрос — практический. Используя опыт составления программы, с помощью которой отыскивается наибольший общий делитель нескольких чисел, составьте программу для нахождения наименьшего общего кратного нескольких чисел.



Этюд 9. УЧЕНИК И КОМПЬЮТЕР

Говорят, что благодаря всеобщей компьютеризации дети разучатся считать в уме. Так ли это? Выдержит ли современный школьник, подружившийся с ЭВМ, экзамен на знание таблицы умножения?

Между прочим, компьютер сам может проверить это, провести такой экзамен с учеником, а заодно потренировать его в устном счете. Раз компьютеры считаются виновниками плохого знания таблицы умножения, то пусть они и исправляют это положение.

На дисплее появляется вопрос типа: «5 помножить на 9 = ?». Экзаменуемый должен быстро ответить — набрать на клавиатуре произведение. Ответ правильный — на дисплее загорается: «Верно!». Ответ неправильный — соответственно и текст на дисплее другой: «Неверно! Ответ такой-то».

Сравните это описание с программой на Бейсике, по которой ведется такой экзамен (рис. 2.3).

Удивлены, не правда ли? Трудно поверить, что для столь несложного экзамена требуется такая длинная программа. Пятнадцать строк! Операторов — не сосчитать! Но если вы разберетесь в этих строках, то увидите, что только малая их часть совершает действия, названные в описании экзамена. Остальные строки нужны для того, чтобы сделать процедуру экзамена удобной и, я бы сказал, приятной для экзаменуемого.

Слишком высокая плата за удобства, скажете вы? Однако вы сочтете ее вполне нормальной, если сами попробуете составлять обучающие и экзаменующие программы. Или хотя

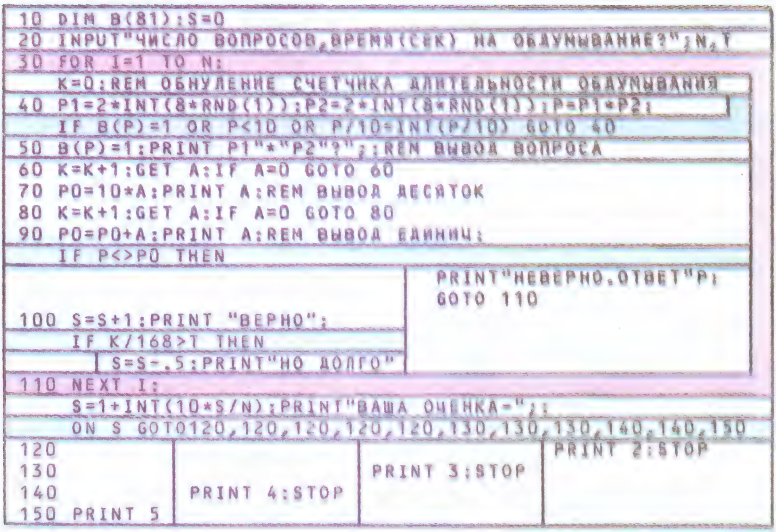


Рис. 2.3. Программа «Таблица умножения»

бы разберете только что описанную — ту, что экзаменует на знание таблицы умножения.

Прежде всего почему бы не предоставить экзаменуемому право самому определить число примеров N и время T на обдумывание каждого из них? Запрос этих величин запрограммирован в строке 20:

20 INPUT «ЧИСЛО ВОПРОСОВ, ВРЕМЯ (СЕК) НА ОБДУМЫВАНИЕ?»;N, T

После того как экзамен закончится, надо выставить оценку экзаменуемому. Значит, следует организовать счетчик верных ответов S, задать начальное нулевое значение этой величины. Это делается в строке 10. И еще стоит позаботиться о том, чтобы один и тот же пример не задавался дважды. Для этого в той же строке 10 вводится массив B(P). Его индекс P — это ответ на очередной пример. Если такой пример встретится хотя бы раз, B(P) получит значение 1. До этого каждый элемент B(P) был равен нулю: начальные нулевые значения автоматически придаются элементам массива при его введении оператором DIM. Проверка B(P) на равенство единице не позволит повторить уже задававшийся раньше пример. Всего таких примеров $9 * 9 = 81$. Это число укажем в скобках в операторе DIM вслед за именем массива B:

10 DIM B(81): S = 0

А теперь начнем задавать экзаменуемому один пример за другим. На язык программирования выражение «один за дру-

гим», как известно, переводится словом «цикл». Пишем заголовок цикла с параметром I:

```
30 FOR I = 1 TO N
```

Прежде чем придумывать очередной пример, организуем счетчик времени, которое экзаменуемый затратит на ответ. Для этого припишем к начальной строке цикла соответствующий оператор:

```
30 FOR I = 1 TO N:K = 0
```

Оба сомножителя машина пусть придумывает с помощью датчика случайных чисел RND(1). Сомножители меньше двух вряд ли понадобятся — слишком простыми будут тогда примеры. Напишем для каждого сомножителя такую формулу: 2 + INT(8 * RND(1)). Сразу же вычислим верное произведение. Если оно меньше или кратно 10, отбросим пример как чересчур легкий. И если он уже задавался раньше, тоже отбросим его:

```
40 P1 = 2 + INT(8 * RND(1)):P2 = 2 + INT(8 * RND(1)):P = P1 * P2:IF B(P) = 1
OR P < 10 OR P/10 = INT(P/10) GOTO 40
```

На следующую строку попадем, если проверки выдержаны. Присваиваем B(P) значение 1, чтобы больше не задавать такого же примера, и выводим на дисплей вопрос:

```
50 B(P) = 1:PRINT P1 «*» P2 « = ?»;
```

Без удобств такая программа заняла бы в структурной диаграмме всего две строки (задумывание сомножителей и выведение их на дисплей), с удобствами — пять.

Пример задан — начинается отсчет времени, за которое экзаменуемый даст свой ответ. Компьютер прибавляет к переменной K единицу за единицей. Чтобы этот счет прервался тотчас же, как только начнется ввод ответа, употребим оператор GET. Он подобен оператору INPUT, но в отличие от него не прерывает выполнение программы. Если в момент выполнения машиной оператора GET будет нажата какая-нибудь цифровая клавиша от 1 до 9, то переменная, стоящая за словом GET, примет соответствующее ненулевое значение. В противном случае (не нажата никакая клавиша или нажата клавиша, отличная от указанных выше) переменная, стоящая за словом GET, примет нулевое значение.

Оператор GET позволяет машине выполнять два действия одновременно: вести счет по программе и как бы прислушиваться к человеку, чтобы изменить свои действия тотчас, как только он нажмет цифровую клавишу. Так, кстати, поступает в диалоге любой вежливый человек, который, ведя свой монолог, следит за собеседником и при первой необходимости дает ему возможность вставить свою реплику или уточняющий вопрос.

Вводить ответ на вопрос машины при проверке таблицы умножения придется в две стадии — ведь мы уже заранее отсеяли слишком простые примеры, а у всех оставшихся ответы двузначные. Экзаменуемый, естественно, сначала будет вводить

число десятков, а потом единиц. Приспособим нашу программу к такому вводу:

```
60 K = K + 1:GET A:IF A = 0 GOTO 60
70 P0 = 10 * A:PRINT A
80 K = K + 1:GET A:IF A = 0 GOTO 80
90 P0 = P0 + A:PRINT A
```

Четыре строчки. А, казалось, хватило бы одной — INPUT P0. Но такой оператор в нашей программе применить нельзя. Он требует нажать по окончании ввода клавишу ВК, и расходуемое на это время исказит отсчет срока, потраченного экзаменуемым на ответ.

Ответ экзаменуемого введен в машину. Проверку ответа и оценивающие реплики машины (ВЕРНО, НЕВЕРНО) запрограммировать нетрудно. Но опять-таки вопрос: почему бы, если экзаменуемый медлит с ответом, не упрекнуть его за это и слегка не наказать? Скажем, скинув ему полбалла? А потом можно задавать следующий пример:

```
 90 P0 = P0 + A:PRINT A:IF P<>P0 THEN
    PRINT «НЕВЕРНО. ОТВЕТ» P:GOTO 110
100 S = S + 1:PRINT «ВЕРНО»:IF K/168>T THEN
    S = S - .5:PRINT «НО ДОЛГО»
110 NEXT I
```

Теперь делим общее количество набранных баллов S на число решенных примеров N и выставляем оценку.

Но как ее выставлять? У разных экзаменаторов разные понятия о «пятерке» и «двойке», о гранях, отделяющих одну оценку от другой. Как запрограммировать такое выставление оценок, при котором можно было бы изменять цену той или иной доли верных ответов?

В Бейсике есть очень подходящий для такой цели оператор. Он называется *оператором выбора*. С его помощью выбирается номер строки, начиная с которого продолжится выполнение программы. И выбирается этот номер, смотря по значению переменной, указанной в операторе, — обозначим ее буквой S.

Вот как выглядит оператор выбора:

```
ON S GOTO . . .
```

А дальше перечисляются номера строк, на которые следует передавать управление. Если S равно единице, то управление передается на строку, номер которой указан первым после слова GOTO, если двойке — на строку, номер которой указан вторым, и т. д. Если же S превышает количество номеров, указанных после слова GOTO, то оператор выбора игнорируется и выполняются операторы, записанные после него. Есть и такие версии, где счет в этом случае прерывается и на дисплей выдается сообщение об ошибке.

В некоторых версиях Бейсика значение переменной, в зависимости от которого производится выбор, может быть и не целым — от него оператор выбора сначала берет целую часть.

Чтобы наша программа была пригодной для всех версий, лучше сделать переменную S целой. Например, так:

```
110 NEXT I:S = 1 + INT(10 * S/N)
```

В таком случае S будет принимать значения от 1 до 11. Итак, одиннадцать градаций. Достаточно для того, чтобы выразить многие нюансы оценок — и «три с плюсом», и «пять с минусом», и т. д. Но ради четкости лучше придерживаться строгой пятибалльной системы. И считать, например, что при S от 1 до 5 включительно ставится «двойка», от 6 до 8 — «тройка», от 9 до 10 — «четверка», а при равном 11 — «пятерка».

Оценка пусть ставится в виде цифры, выводимой на дисплей. Но лучше, конечно, сопроводить ее словами: ВАША ОЦЕНКА — 2 или ВАША ОЦЕНКА — 5.

Слова можно вывести заранее, в той же строке 110. Там же можно поставить оператор выбора, который передаст управление на одну из четырех следующих строк, со 120-й по 150-ю, где на дисплей выводятся собственно оценки. Каждая из этих строк заканчивается оператором STOP, прекращающим выполнение программы: экзамен закончен.

Тот, кто разберет заключительный фрагмент программы, убедится, что он полностью соответствует приведенному выше описанию.

Вот так и набегают строчки в программе. Пугаться их обилия не надо. Разобраться в них несложно. А раз так, пусть их будет побольше — лишь бы удобнее было работать с программой.

Почему учтены не все числа?

Тот, кто по нашему совету внимательно выискивает в каждом этюде поводы для самостоятельных рассуждений, конечно же, обратил внимание на фразу из описания программы-экзаменатора: «Если оно (произведение) меньше или кратно десяти, отбросим пример как чересчур легкий». Кратность десяти вряд ли является безоговорочным признаком легкости. Здесь, очевидно, дело в другом.

В чем же? Попробуйте сами ответить на этот вопрос. В поисках ответа обратите внимание на условный оператор в «замкнутых на себя» строках 60 и 80 (см. рис. 2.3).

Таймер компьютера

В строке 100 разобранной нами программы (см. рис. 2.3) встречается дробь K/168. Откуда столь странный знаменатель? Составитель программы определил, что в его машине за одну секунду оператор присваивания в строках 60 и 80 успевает 168 раз прибавить единицу к величине K. Поэтому дробь K/168 выражает в секундах время, затраченное экзаменуемым на обдумывание ответа. Разумеется, точное значение знаменателя нужно подбирать для каждой конкретной машины особо.

Необходимость в таком счетчике, к которому прибегнул составитель программы, конечно, отпала бы, если бы ЭВМ была снабжена часами, показания которых можно было бы считывать специальным оператором. Подобные часы (даже с календарем) есть у многих современных персональных компьютеров, и нам еще доведется поговорить о таком удобном счете времени.

STOP и END

Мало кто из знатоков Бейсика задумывается над тем, чем различаются операторы-близнецы STOP и END. Оба прерывают выполнение программы, любой из них, как правило, можно поставить в любом ее месте. Однако после прерывания по команде STOP можно продолжить выполнение программы, нажав клавишу CONTINUE, если она имеется. Оператор END такой возможности не предоставляет.

Работа компьютера по программе прекращается и в том случае, если вслед за выполненным оператором в ней нет больше ни одной строки.

Этюд 10. ЕСЛИ БЫ Я РАБОТАЛ КАССИРОМ В КИНОТЕАТРЕ

— Вы видели мои рисунки. Вы смотрели мои программы. Вам они показались довольно удачными.

— Ну, что касается рисунков, то тут я ценитель небольшой. А программы у Вас, действительно, получаются вполне грамотные.

— Спасибо. Вот потому я и подумываю о том, что теперь уже имею право самостоятельно выбирать темы для программирования. Я хочу научить компьютер рисованию. Чувствую, что это гораздо труднее, чем решать на нем математические задачи. Мне даже кажется, что это абсолютно разные вещи: задачи, уравнения, вычисления — одно, а линии, фигуры, рисование — совсем другое.

— Вы не совсем правы. Было время, когда математики не скупились на образные, можно даже сказать, живописные выражения. Вот эту кривую (рис. 2.4,б), которую сегодня называют декартовым листом, сам Декарт именовал листом жасмина. Бернулли, восхищенный изящными очертаниями одной математической кривой, назвал ее лемнискатой, что значит «увитая лентами» (рис. 2.4,а). А вот за этой кривой прочно утвердилось название «локон Марии Аньези» — по имени женщины-математика, которая ее впервые исследовала (рис. 2.4,в).

— Эта линия, пожалуй, самая простая из всех, которые вы мне показывали. Скажите, сможет ли компьютер нарисовать ее на дисплее?

— Сначала давайте подумаем, как ее рисовать на листе бумаги. Вот ее уравнение: $y = 1/(1 + x^2)$. Нарисуем две оси

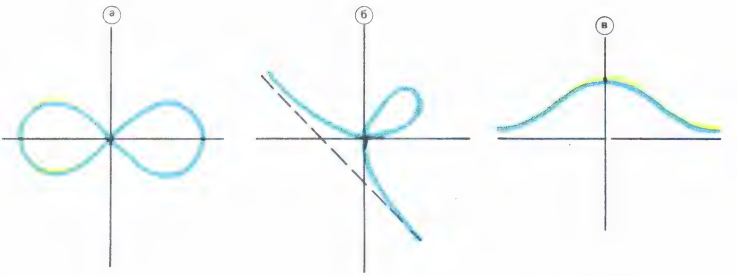


Рис. 2.4. Замечательные кривые: а — лемниската; б — декартов лист; в — локон Марии Аньези

координат. Лист, таким образом, превратится в координатную плоскость, где каждая точка имеет свою абсциссу x и свою ординату y. Возьмем произвольную точку и подставим ее координаты x и y в уравнение кривой. Если равенство выполняется при таких x и y — отметим эту точку на плоскости маленьким кружком. Не выполняется — оставляем эту точку без окраски. Все проставленные нами кружки сольются в график кривой. Вот вам и рецепт рисования. Введите в память компьютера данные об окраске каждой точки дисплея — и на нем возникнет задуманная вами кривая.

— Но точек на плоскости — бесконечное множество. А память компьютера ограничена, как бы ни была велика.

— Согласен. Давайте искать выход из этого противоречия. Расчертим дисплей на квадратики и каждый будем заливать

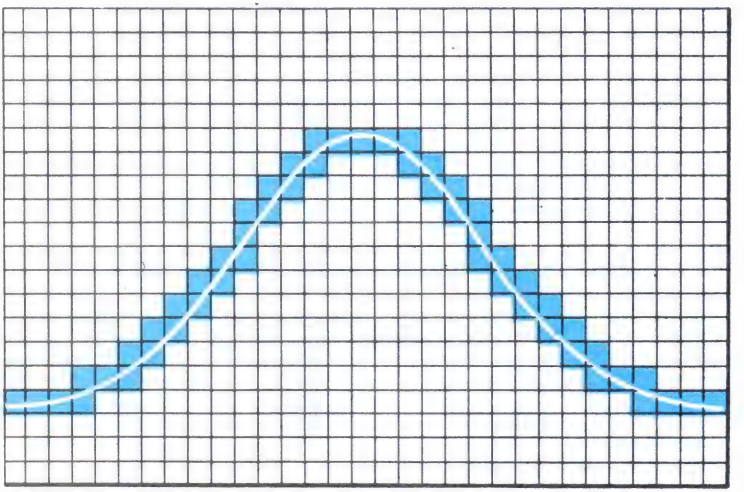


Рис. 2.5. Локон Марии Аньези, изображенный в «клеточной» манере

своим тоном. Если в каком-то квадратике окажется хотя бы одна точка с координатами, удовлетворяющими уравнению кривой, закрасим его весь темным цветом. В результате локон Марии Аньези будет выглядеть так, как его изобразили на рис. 2.5.

— Выглядит довольно грубовато.

— Хотите больше изящества — мельче дробите экран, берите больше клеток. Вспомните, как тонко передаются линии и контуры на экране телевизора. А ведь там принцип изображения почти тот же, только клетки очень маленькие, измеряемые долями миллиметра.

— Представляю, сколько их на экране. Если вводить в память компьютера данные об окраске каждой клетки — хватит ли на это памяти?

— Резонный вопрос. Я сейчас покажу вам, что иногда для такого рисования требуется не так уже много клеток. Вот смотрите — рис. 2.6. Он выполнен все в той же «клеточной»

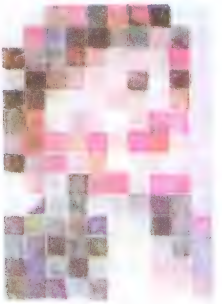


Рис. 2.6. Портрет человека, выполненный в «клеточной» манере

технике, но клетки здесь гораздо крупнее. Узнаете, кто тут изображен?

— Человек ... с пышной шевелюрой ... в черном костюме ...

— А все поле рисунка содержит лишь 140 квадратов: десять — по горизонтали и четырнадцать — по вертикали. Совсем немного! И тем не менее рисунки получаются вполне узнаваемые.

— Не совсем ясно, правда, как программировать такие рисунки.

— Давайте разберемся для начала в том, как запрограммировать хотя бы один ряд этого разграфленного поля. Все клетки можно пронумеровать: 1, 2, 3, ... А потом остается ввести в память компьютера сведения об окраске каждой клетки: A(1), A(2), A(3), ...

— Получается массив!

— Совершенно верно! Получается массив A(J), в котором J-й элемент выражает интенсивность тона той клетки, которая

в нашем ряду стоит на J-м месте. Следующий шаг, который нам предстоит сделать, — построить массив, каждый элемент которого выражает интенсивность тона клетки, стоящей на определенном месте в определенном ряду.

— Ряд, место . . . Прямо, как в кинотеатре!

— В кинотеатре, говорите? А это идея! Давайте составим программу, которая помогла бы кассиру кинотеатра продавать билеты. Так мы и полезное дело сделаем, и лучше поймем, как научить компьютер рисованию. Идет?

— Идет. Но как же все-таки построить массив для раскраски клеток, с номерами ряда и места?

— Да очень просто (рис. 2.7). Пишем A(I,J). Когда в обозначении массива в скобках указываются два индекса через запятую, массив называется двумерным. Пусть произвольный элемент A(I,J) из этого массива выражает окраску клетки, которая расположена в I-м ряду на J-м месте.

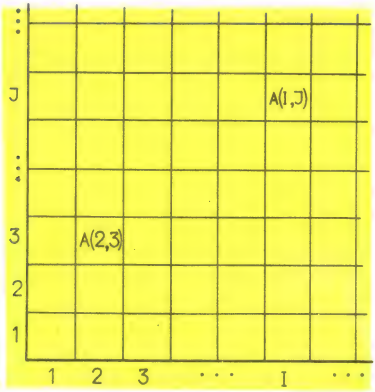


Рис. 2.7. Структура двумерного массива

— В самом деле, очень просто.

— Тогда давайте займемся задачей о продаже билетов в кинотеатр. Подумаем, как приспособить массив A(I,J) для программы, помогающей кассиру в его работе. Пусть элементы нашего массива принимают лишь два значения: либо 0, если соответствующее место свободно, либо 1, если занято. До начала продажи билетов, когда программа только введена в компьютер и еще не работала, элементы всех используемых в ней массивов автоматически полагаются равными нулю. Это мы учтем при составлении программы. Пусть по ней на дисплее будет нарисован план зрительного зала, где на свободном месте поставлена буква С, на занятом — буква З. Я думаю, что такую программу мы составим без труда (рис. 2.8). В первой ее строке зададим число рядов R1 и число мест в каждом ряду M1. Например:

```
10 R1 = 15:M1 = 10
```

```
10 R1=15:M1=10:DIM A(15,10)
20 FOR K=1 TO R1*M1:REM ЗАГОЛОВОК ЦИКЛА ПО ЗРИТЕЛЯМ
30 PRINT "  ";:
   FOR M=1 TO M1:PRINTUSING "###",M;:NEXT M:
   PRINT
40 FOR R=1 TO R1:
   PRINTUSING "##",R;" ";
50 FOR M=1 TO M1:
   IF A(R,M)=1 THEN 60:PRINT "C  ";:GOTO 70
60 PRINT "З  ";
70 NEXT M:PRINT :REM КОНЕЦ ЦИКЛА ПО МЕСТАМ В РЯДУ
80 NEXT R:REM КОНЕЦ ЦИКЛА ПО РЯДАМ В ЗАЛЕ
90 INPUT "ЖЕЛАЕМЫЙ РЯД,МЕСТО",R,M:
   IF R>R1 OR M>M1 THEN 100:
   IF A(R,M)=0 THEN 110:
   PRINT "ЭТО МЕСТО УЖЕ ЗАНЯТО.":GOTO 90
100 PRINT "ТАКОГО МЕСТА В ЗАЛЕ НЕТ.":GOTO 90
110 A(R,M)=1:IF R<4 OR R>R1-3 THEN 120:S=50:GOTO 130
120 S=40:REM ДЕШЕВЫЕ БИЛЕТЫ
130 PRINT "С ВАС";S;" КОПЕЕК"
140 NEXT K:
    PRINT "ВСЕ БИЛЕТЫ ПРОДАНЫ"
```

Рис. 2.8. Программа «Кассир в кинотеатре»

И тут же введем наш двумерный массив.

— А как его ввести? С помощью какого оператора?

— Того же, с помощью которого мы вводили одномерные массивы:

DIM A(15, 10)

На первом месте в скобках — максимальное значение первого индекса, на втором — второго. Припишем этот оператор в конце строки 10, а в следующей строке устроим счетчик проданных билетов:

20 FOR K = 1 TO R1 * M1

Последующими строками нарисуем план зрительного зала (рис. 2.9). Будем перебирать с помощью индекса R ряд за рядом, а в каждом ряду с помощью индекса M место за местом и печатать букву С, если A(R,M) = 0, или букву З, если A(R,M) = 1.

— Мне кажется, что я сам смогу написать этот кусочек программы.

— Попробуйте. Только учтите: мало нарисовать на дисплее табличку из букв С и З. По верхнему ее краю должны стоять номера мест, по левому — номера рядов.

— Да-да! Сначала, в самой первой строке я напечатаю номера мест:

30 FOR M = 1 TO M1:PRINT M;: NEXT M

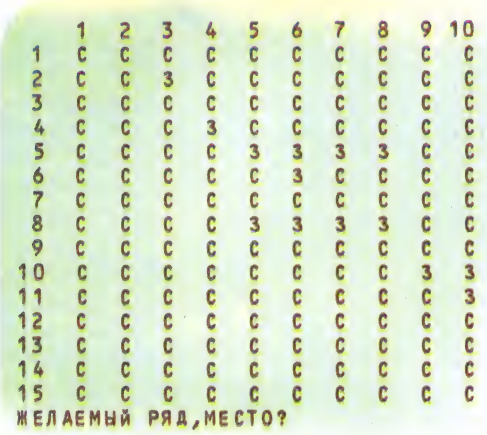


Рис. 2.9. План зрительного зала в кинотеатре, отображенный на экране дисплея

— Не очень-то красивой получится у вас эта вереница номеров: все цифры почти слипнутся. Давайте печатать их с пробелами. Для этой цели могу вам посоветовать хороший оператор. Пишите:

```
PRINT USING«# # #»,M
```

— Как он выполняется, этот оператор?

— При печати номер M, однозначный или двузначный, заменит один или два знака диеза справа, а оставшиеся диезы заменятся пробелами. Так, проставляя нужное число диезов, можно задавать подходящий *формат печати*. Кстати, еще совет: оставьте два пробела с левого края — под ними в следующих строчках будут печататься номера рядов.

— Пробел — это ведь пустое место в кавычках . . .

```
30 PRINT«  »;:FOR M = 1 TO M1:PRINTUSING «# # #»,M;:NEXT M
```

Дальше я продолжу сам: начало цикла FOR R = 1 TO R1 . . . номер ряда . . . здесь я опять использую ваше PRINT USING . . . пробел . . . номера мест в ряду — тоже циклом . . . буква — и два пробела за нею . . .

```
40 FOR R = 1 TO R1:PRINTUSING «# # #»,R:«  »;:
50 FOR M = 1 TO M1:IF A(R,M) = 1 THEN 60:PRINT «C»;:GOTO 70
60 PRINT «З»;
70 NEXT M
80 NEXT R
```

Правильно?

— Не совсем. У вас же все буквы будут напечатаны в один ряд. А вам, перебрав в каком-то ряду все места, т. е. перебрав все значения M от 1 до M1, надо перейти на новую строчку.

— Написать в конце строки 70 команду BK — возврат каретки?

— Такой команды нет. Есть только клавиша с таким обозначением. Но нужное вам действие можно совершить, если в строке 70 припишите оператор PRINT без всяких последующих указаний. Если в нем не указано, что печатать, он означает лишь то, что дальнейшая информация должна выводиться на дисплей с новой строки, с самого ее начала. Иными словами, он совершает то же самое, что делает машинистка, нажимая рычаг для перевода каретки. Подумайте: где еще следует приписать точно такой же оператор?

— В самом начале строки 40, чтобы «перевести каретку» после того, как в строке 30 были напечатаны номера мест.

— Совершенно верно. Ну вот, считайте, что план зрительного зала на дисплее. Глядя на него, кассир быстро определит, свободно ли место, которое у него просит зритель.

— А почему бы не поручить машине самой отвечать на запрос зрителя?

— Отличная мысль! Допустим, зритель назвал желаемый ряд и место в нем. Кассир вводит эти данные в машину. Напишем соответствующую строку программы:

```
90 INPUT «ЖЕЛАЕМЫЙ РЯД, МЕСТО», R,M
```

— Хорошо бы тут же устроить проверку: не ошибся ли зритель? Вдруг он назвал ряд, которого нет в зале? Или место, которого нет в ряду?

— Или место, которое уже занято. Сейчас мы все эти запреты учтем:

```
90 INPUT «ЖЕЛАЕМЫЙ РЯД, МЕСТО»,R,M:
   IFR>R1 OR M>M1 THEN 100:IFA(R,M) = 0 THEN 110:
   PRINT «ЭТО МЕСТО УЖЕ ЗАНЯТО.»:GOTO 90
100 PRINT «ТАКОГО МЕСТА В ЗАЛЕ НЕТ.»:GOTO 90
```

Изучите эти две строки. На строку 110 машина перейдет лишь в том случае, если заказанное зрителем место есть в зале и не занято.

— Тогда выводим на дисплей: «С ВАС 50 КОПЕЕК».

— Нет, постойте! Не все так просто! Ведь в первых трех и в последних двух рядах билеты на 10 копеек дешевле. И еще надо сделать так, чтобы кассир не продал два билета на одно место!

— Да, с этого бы надо начать строку 110 — присвоить элементу A(R,M) значение 1. А сообщение на дисплее запрограммировать так: «С ВАС»; S; «КОПЕЕК». Значение S, равное 50 или 40, пусть определяет программа в зависимости от значения R. Сейчас, сейчас ... Ну вот, пожалуй, так:

```
110 A(R,M) = 1:IF R<4 OR R>R1-3 THEN 120:S = 50:GOTO 130
120 S = 40
130 PRINT «С ВАС»;S;«КОПЕЕК»
```

— Отлично! Ну что ж, все операции по продаже заказанного клиентом билета запрограммированы. Что теперь вы предла-

гаете написать в нашей программе? Задумались? Просмотрите всю ее целиком. Видите: в строке 20 написан заголовок цикла FOR K = 1 TO R1 * M1, а завершающего цикл оператора NEXT K у вас нигде нет.

— И вправду нет! Поставим его сейчас, в строке 140?

— Давайте, поставим. Посмотрите, как тогда будет выполняться цикл. «Прокрутите» его, как говорят программисты. Совершаемые в нем действия представляйте, так сказать, крупными мазками, без той детализации, с которой мы их описывали при составлении программы. Начали?

— Начали. Строка 20, заголовок цикла. Параметр K получает начальное значение 1. Строки 30—80: на дисплее рисуется план зрительного зала. Все элементы A(R,M) еще равны нулю, так что на всех местах зала стоят буквы С — «свободно». Строка 90 — оператор INPUT. Выполнение программы приостанавливается. На дисплее — запрос о желаемом ряде и месте. Вводим эту информацию, нажимаем клавишу ВК. Следующие операторы строки 90 и далее до строки 130: элемент A(R,M), соответствующий заказанному месту, получает значение 1; на дисплее — просьба уплатить стоимость билета; оператор NEXT K прибавляет единицу к счетчику проданных билетов, и цикл выполняется снова, со строки 30. Опять на дисплее рисуется план зрительного зала, только теперь на месте, на которое продан билет, стоит буква З — «занято». И так после каждого запроса.

— Во время остановки, диктуемой в строке 90 оператором INPUT, кассир получит со зрителя деньги, выпишет билет, получит запрос от следующего зрителя и передаст его машине... Так цикл будет выполнен R1 * M1 раз. А когда произойдет выход из цикла, на дисплее появится знакомая всем надпись: «ВСЕ БИЛЕТЫ ПРОДАНЫ»

— Все ли мы учли? Дайте-ка я еще раз посмотрю программу. 15 рядов по 10 мест в каждом. Но ведь так не в любом зале.

— Конечно, число рядов и мест в зале может быть и не такое. Для этого нужно соответствующим образом изменить строку 10, присвоить другие значения переменным R1 и M1, потом вписать те же самые значения в скобки, стоящие вслед за словом DIM. Конечно, вы, как художник, могли бы еще посетовать на то, что наша программа годится лишь для прямоугольных залов. Бывают ведь и залы, где в разных рядах число мест неодинаково.

— Если бы вы лучше представляли работу художника, вы бы не были столь боязливы в своих опасениях. Когда художник только набрасывает композицию рисунка, он часто заключает изображаемый предмет в условные прямоугольные рамки. А потом дорабатывает эскизный контур до нужной формы. Наша

программа — это только набросок, не правда ли? Если кто-то хочет доработать ее для зала, где число мест в ряду зависит от номера ряда, — пусть попробует.

— Я бы для таких любителей подсказал еще одну возможность обобщения. Массив A, хранящий информацию о наличии свободных мест, может быть и более чем двумерный. Он может информировать о различных сеансах в разные дни во всех кинотеатрах города. Например, если A(R,M,I,J,K) = 0, то это значит, что в R-м ряду M-е место на I-й сеанс в J-й день в K-м кинотеатре города пока еще не продано. Так можно создать автоматизированную систему управления (АСУ) зрелищными заведениями всего города. Впрочем, наша программа-набросок — тоже не безделушка. Если бы я был кассиром в кинотеатре, я установил бы на своем рабочем столе персональный компьютер, ввел в него нашу программу и не знал бы никаких хлопот. А еще лучше было бы снабдить компьютер монетоприемником, как в автомате для продажи билетов на электричку. Тогда можно обойтись и без кассира: билет с указанием места и ряда напечатает принтер.

Программные блоки

Многое из того, что рассказывает программист, художнику кажется откровением. Между тем основные положения программирования по существу очень родственны правилам, которыми руководствуется человек в любой своей деятельности, в том числе в рисовании. Изложенные непривычным языком в руководствах для начинающих программистов, они выглядят странными, но если вникнуть в них, они покажутся естественными. Далее, опираясь на них, легко усвоить всю программистскую грамоту.

Создание картины художник обычно начинает с того, что разрабатывает ее композицию — располагает на плоскости холста ее основные крупные фрагменты. Каждый фрагмент он набрасывает опять-таки по частям, но более мелким, те в свою очередь состоят из частей поменьше — и так до мельчайшего штриха.

В этом описании программист усмотрел бы одно из основных правил структурного программирования: разлагать каждый блок исходного алгоритма на блоки все более мелкие и простые, вплоть до отдельных операторов.

Следуя этому правилу, старайтесь уже при составлении программы мыслить ее в виде системы крупных блоков, соединенных между собой логическими взаимосвязями, а внутри себя имеющих ту же блочную структуру. Умейте прослеживать эту иерархию структур сверху донизу, беря ее во все более детальных планах — до строк, до операторов.

Размерность массивов

В большинстве версий Бейсика максимальная размерность массива равна двум или четырем. Бóльшая размерность, как правило, не нужна при решении задач, для которых создавался Бейсик. Кроме того, в массиве, элементы которого имеют слишком много индексов, несложно и запутаться. Не машине, конечно, а человеку, создающему для нее программу.

Списки данных

Как учесть в программе непрямоугольные очертания зала? Как принять во внимание наличие неудобных мест, билеты на которые продаются в последнюю очередь, бронирование мест и прочие нюансы?

В приведенном варианте программы элементы массива A(R,M) принимали только два значения — нуль (место свободно) и единица (билет на это место продан). Однако ничто не запрещает присваивать его элементам и другие значения. Пусть равенство A(R,M) = 2 означает, что такого места в зале нет; A(R,M) = 3 — что это место забронировано, и билет на него поступает в свободную продажу за 30 минут до начала сеанса. В этом случае перед началом продажи билетов массив A(R,M) должен хранить не нули (их присваивал его элементам оператор DIM), а другие значения, определяющие начальный статус места. Как же присвоить их соответствующим элементам массива? Можно, конечно, вставить в программу операторы присваивания, допустим A(1,1) = 2; A(1,2) = 0; . . . ; A(8,5) = 3; . . . ; A(15,10) = 2, но уж очень длинной получится такая конструкция.

Есть другой, более удобный способ — считывать данные из списка с помощью тройки операторов: READ (читать), DATA (данные) и RESTORE (восстановить). Для этого в конце программы (можно и в начале, если бы там было свободное место) отобразим начальное состояние зрительного зала в виде *списка данных*:

```
500 DATA 2,0,0,0,0,0,0,0,0,2:REM 1 РЯД НЕПОЛНЫЙ
510 DATA 0,0,0,0,0,0,0,0,0,0:REM 2 РЯД ПОЛНЫЙ
520 . . .
570 DATA 3,3,3,3,3,3,3,3,3,3:REM 8 РЯД БРОНЬ
580 . . .
640 DATA 0,0,0,0,2,2,0,0,0,0:REM 15 РЯД НЕПОЛНЫЙ
```

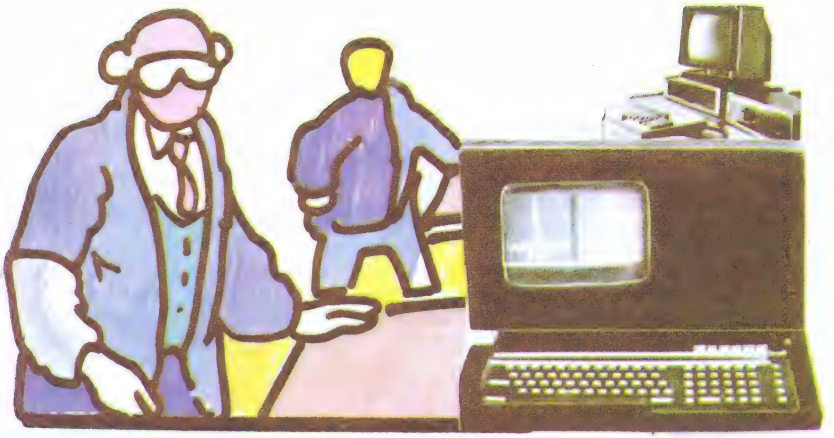
Теперь, чтобы считать эти данные из списка и заполнить ими массив A(R,M), достаточно воспользоваться оператором READ, заключенным в двойной цикл:

```
FOR R = 1 TO R1:FOR M = 1 TO M1:READ A(R,M):NEXT M:NEXT R
```

Присвоив элементу A(1,1) значение 2, указанное в списке первым, оператор READ при следующем обращении к нему выберет из списка второе значение, что даст A(1,2) = 0, при следующем — третье и т. д. Исчерпав список, помещенный в строке 500, оператор READ при дальнейших обращениях к нему будет искать списки данных, отмеченные словом DATA, в следующих строках программы и опять будет последовательно извлекать все перечисленные там значения.

Если же возникнет необходимость считать данные из списка еще раз, то установить оператор считывания READ на первое число в первом из имеющихся в программе списков поможет оператор RESTORE.

В некоторых версиях Бейсика вслед за словом RESTORE можно указывать число, определяющее, с какой позиции должно начаться новое считывание значений из списка DATA при обращении к нему оператором READ.



Этюд 11. СКВОЗЬ ТУМАН К ВЕРШИНЕ ХОЛМА

Это сейчас Бейсику выучиться просто — и лекции читаются, и занятия ведутся, и книги издаются, и статьи в научно-популярных журналах печатаются. А в то время, когда в моей лаборатории появилась «Искра 226», этого почти не было. Хорошо, что ребята в моем отделе подобрались энергичные: быстро организовали краткий курс лекций по Бейсику для всех сотрудников; каждый слушатель потом сдал экзамен, самостоятельно решил на машине пару простеньких примеров — составить таблицу значений функций, найти ее корень на отрезке. . .

«Николай Петрович, — докладывают мне, — можете считать, что Бейсик мы освоили!».

«Нет, — говорю, — рановато вы меня в этом уверяете! Знаем мы ваши скоростные методы: сегодня лекцию прослушал, завтра экзамен сдал, послезавтра все из головы вылетело! Так ведь? Так».

И кому бы я с тех пор ни давал какую-то задачу, всегда требовал, чтобы мне представляли программу ее решения для нашей «Искры». За глаза меня, конечно, поругивали. А я, знай, требую свое. Зато сейчас могу поручиться: неграмотных (в «компьютерном» смысле) у меня в лаборатории нет. А есть, между прочим, и такие, что за их программами ко мне из других лабораторий обращаются, потому что многие наши задачи по этим «самодеятельным» программам решаются лучше, чем по тем, что описаны в справочниках по программированию.

Нам часто приходится решать задачи на поиск оптимума, а в математическом смысле — на поиск экстремума функции нескольких переменных, чаще всего — ее максимума. (Впрочем, если у функции поменять знак на противоположный, то будет

```
10 N=...:REM N-ЧИСЛО АРГУМЕНТОВ:DIM X(N):GOTO 30
20 Y=FN(X(1),...,X(N)):RETURN:REM ЗДЕСЬ НУЖНО ЗАПИСАТЬ
   ФУНКЦИЮ ИЛИ ПРОЦЕДУРУ ОПРЕДЕЛЕНИЯ ЗНАЧЕНИЯ Y
30 FOR I=1 TO N:PRINT"X"I"НАЧ";:INPUT X(I):NEXT I
40 GOSUB 20:YMAX=Y:INPUT"НАЧ.ШАГ,ТОЧНОСТЬ?";D,DMIN
50 IF D<DMIN GOTO 170:REM НАЧАЛО ВНЕШНЕЙ ИТЕРАЦИИ
60 P=0:REM ОБНУЛЕНИЕ ПРИЗНАКА
70 FOR I=1 TO N:REM ПЕРЕБОР ПЕРЕМЕННЫХ
80 FOR X=-D TO D STEP 2*D:REM ДВА ШАГА
90 X(I)=X(I)+X:GOSUB 20
100 IF Y>YMAX THEN
        P=X:J=I:YMAX=Y
110 X(I)=X(I)-X
120 NEXT X
130 NEXT I
140 X(J)=X(J)+P:REM ПЕРЕХОД К НОВОЙ ТОЧКЕ
150 IF P<>0 GOTO 60:REM КОНЕЦ ВЛОЖЕННОЙ ИТЕРАЦИИ
160 D=D/2:GOTO 50
170 FOR I=1 TO N:PRINT"X"I"ОПТ="X(I):NEXT I:REM ОТВЕТ
```

Рис. 2.10. Программа «Максимум»

вестись поиск минимума.) Вообще поиск экстремумов функций — одна из самых распространенных задач, решаемых на ЭВМ. Методов для этой цели придумано много — бери любой и программируй. Но в одном методе функцию обязательно надо задавать в явном виде, в другом — вычислять ее частные производные.

Один инженер нашей лаборатории недавно написал программу, которая не требует подобных условий. Она работает так же, как путник восходит на вершину холма, которую не видно из-за густого тумана. Но у путника есть компас, и он решает искать вершину так: ощупывает склон в четырех направлениях — на восток, на запад, на север, на юг, а затем переступает туда, где подъем оказался максимальным. Так он будет действовать до тех пор, пока пробные шаги не покажут понижение рельефа во всех четырех направлениях. Тогда дальнейший поиск пойдет с уменьшенным вдвое шагом. Когда шаг станет меньше заданной точности, достигнутая точка полагается искомой точкой максимума.

Потом, когда мы вместе разобрались в программе, я заметил, что ее бы лучше назвать, как известный танец, — «тустеп», что в переводе значит «два шага». Потому что по этой программе от очередной исследуемой точки по каждой координате делается шаг в отрицательном и шаг в положительном направлении. Координат же в пространстве, где задана функция, может быть сколько угодно, т. е. число аргументов у функции практически неограничено, так что уподоблять ее холму было бы необоснованно.

Программа получилась короткая (рис. 2.10), и я не пожалел времени, чтобы разобраться в ней до строчки.

Строка 10. Ввод числа аргументов функции. Переход на строку 30, с которой и начинается поиск точки максимума.

Строка 20. Вычисление величины Y — значения исследуемой функции. Оно оформлено в виде подпрограммы — это видно по слову RETURN (возвращаться), стоящему в конце строки.

Строка 30. Цикл с параметром. Он выполняется последовательно при всех значениях индекса I, указанных в заголовке цикла. Вводятся координаты начальной точки, с которой начинается пошаговое движение к максимуму, в виде массива X (I).

Строка 40. Оператором GOSUB 20 управление передается на подпрограмму, записанную в строке 20 (полное название оператора — go to subrutine — идти к подпрограмме). Вычисленное там значение функции в начальной точке поиска присваивается переменной YMAX. Далее запрашивается начальный шаг приближения к максимуму D и минимальный шаг DMIN, определяющий точность расчета.

Строка 50. Типичное начало *цикла «пока»*: проверяется условие и, если оно не выполняется, ведутся запрограммированные далее вычисления — вплоть до строки 160, откуда управление снова передается на строку 50, где вновь проверяется то же условие. Когда оно, наконец, выполнено, управление передается на строку 170, где выводится результат вычисления по программе.

Строка 60. Еще один типичный прием программирования — введение признака, по которому совершится выход из цикла. Право на выход из него появляется только в том случае, если смещение по каждой координате как в отрицательном, так и в положительном направлении не вызывает увеличения функции. Сначала в строке 60 признаку P придается нулевое значение. Если при смещении из исследуемой точки в какую-то соседнюю зафиксировано увеличение функции, признаку присваивается ненулевое значение. А в конце цикла поставлена проверка признака P на нуль. Выход из цикла происходит по условию P = 0.

Две следующие строки начинаются со слова FOR. Ясно: один цикл с параметром *вложен* в другой.

Строка 70. Начинается перебор координат от 1-й до N-й. По каждой из них будем смещаться, чтобы проверить, не увеличивается ли в этом направлении функция.

Строка 80. Когда я читал программу первый раз, эта строка удивила меня. FOR X = — D TO D STEP 2 * D. Это значит, что переменная X вначале должна принять значение — D, а в конце + D, возрастает же она с шагом (STEP), равным 2D. Стало быть, для достижения конечного значения D шаг 2D будет прибавлен к начальному значению — D всего один раз.

Не проще ли взять сначала X = — D, а потом X = D? Как это запрограммировать, я сразу придумать не смог, а на долгие раздумья не было времени, и я решил, что цикл здесь вполне уместен. Свою задачу он выполняет — переменная X принимает все те и только те значения, которые нужны согласно алгоритму.

Строка 90. Очередная координата X(I) получает приращение X. Вычисляется значение функции в точке, заданной таким смещением.

Строка 100. Если только что вычисленное значение функции Y не превысило значение YMAX, то переходим на строку 110 и там возвращаем переменной X(I) неискаженное значение. Если же превысило, то заменяем нулевое значение признака P на ненулевое, D или — D, смотря по тому, чему в данный момент равно X. Присваиваем переменной J значение номера координаты, вдоль которой зафиксировано возрастание функции. Наконец, придаем величине YMAX превышающее ее значение Y.

В этом присваивании, можно сказать, вся соль программы. Если рост функции будет отмечен еще вдоль каких-то координат, то в итоге переменная YMAX будет равна наибольшему из значений функции в точках, соседних с исходной. В какой именно из этих точек функция принимает такое значение, мы определим, посмотрев, чему равны переменные J и P.

Строка 110. Зафиксировано возрастание функции в точке, соседней с исходной, или не зафиксировано, координате X(I) все равно возвращается неискаженное значение.

Строка 120. Обязательно испытываем оба приращения координаты X(I) — и отрицательное, и положительное.

Строка 130. Перебираем все координаты. Два слова NEXT в соседних строках 120 и 130 — завершение двух циклов с параметром, один из которых вложен в другой.

Строка 140. Смещаемся в точку наибольшего роста функции.

Строка 150. Если P не равно нулю, значит, рядом с исходной точкой с координатами X(1), X(2), . . . , X(N) нашлась такая, где функция имеет большее значение, т. е. максимум еще не достигнут. Надо продолжать поиск. Переходим для этого на строку 60. Типичная концовка цикла «до»: проверяется условие и, если оно не выполняется, управление передается на следующую строку программы, если выполняется — на первую строку цикла.

Строка 160. Попадаем сюда только в том случае, если ни в одной из соседних точек значение функции не оказалось большим, чем в исходной точке. Значит, точка максимума где-то совсем рядом. Надо уточнить ее положение. Продолжаем ее поиск с уменьшенным вдвое шагом. Заменяем D на D/2 и переходим на строку 50. Там — проверка: D<DMIN ? Если «да»,

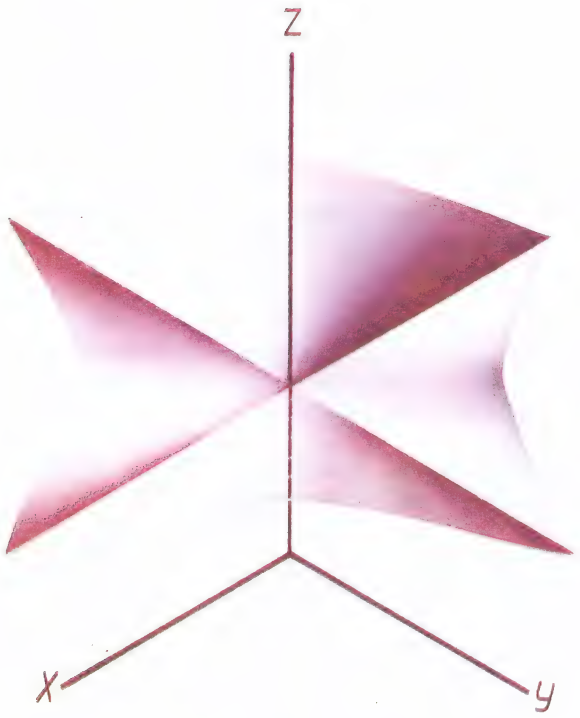


Рис. 2.11. График функции двух переменных, у которой программа «Максимум» ошибочно отыскивает точку максимума в начале координат

то мы находимся в искомой точке максимума. Ее положение определено, разумеется, с точностью DMIN. Переходим на строку 170.

Строка 170. Печатаем координаты точки максимума.

Прошелся я по этой программе еще разок... Хорошая программа! Только для всех ли функций она годится? Взял я лист плотной бумаги, согнул его и положил на стол перед автором программы: вот, мол, функция двух переменных X и Y; на осях координат она имеет постоянные значения, а в секторах возрастает (рис. 2.11). «Ваша, — говорю, — программа, если взять за исходную точку начало координат, не обнаружит роста функции в направлении осей и сделает вывод, что начало — это точка максимума. А ведь никакого максимума тут нет. Верно? Верно. Что вы на это скажете?»

«Для каждого оптимизационного алгоритма, — отвечает он мне, — можно подобрать функцию, которая ему окажется

не по зубам. Так что вы меня своим примером не поразили. К тому же он довольно искусственный. Сколько я с моей программой ни работаю, таких мне еще не встречалось».

Я говорю: «А с теми примерами, что до сих пор были, как она справлялась?» Отвечает, что справлялась отлично. Хотя есть у нее еще и такой недостаток: ищет только локальный максимум.

Договорились, что после первого же случая, когда его программа не отыщет точку локального максимума там, где она заведомо есть, он усовершенствует алгоритм, перепишет программу и представит мне этот усовершенствованный вариант.

Принципы структурного программирования

Разбирая предыдущие программы, внимательный читатель заметил, что все циклы можно разбить на три вида. Все они представлены и прокомментированы в только что описанной программе «Максимум» (см. рис. 2.10).

На структурных диаграммах циклы различаются раскраской: циклы «до» и «пока» помечаются синим цветом, цикл с параметром — красным.

Нередок на структурных диаграммах и зеленый цвет. Им раскрашивается верхняя полоска альтернативной конструкции, содержащая условие. В программе «Максимум» в строке 100 присутствует простейшая из таких конструкций — условный оператор. Здесь левый из прямоугольников, расположенных под зеленой строкой, пуст — при несоблюдении условия никаких действий не совершается. В программе «Таблица умножения» (см. рис. 2.3) в строках 90 — 100 нам встречалась альтернатива в полном ее виде, когда при соблюдении условия выполняются одни действия, а при его несоблюдении — другие. В той же программе «Таблица умножения» в строках 110 — 150 записана альтернативная конструкция наиболее общего вида — так называемый оператор выбора.

Функциональные блоки на структурных диаграммах окраски не имеют. И если они следуют друг за другом, диаграмма выглядит бесцветной.

Итак, мы перечислили все управляющие конструкции, допустимые принципами структурного программирования: следование, повторение (этим термином объединяются все виды циклов), выбор (во всех его разновидностях, включая условный оператор и альтернативу). На рис. 2.12,*а* — 2.12,*е* показано, как эти элементы изображаются на структурных диаграммах и блок-схемах. При всей их немногочисленности из этих элементов удается строить весьма сложные алгоритмические сооружения. Дело в том, что их можно разнообразно комбинировать. Например, циклы можно вкладывать друг в друга (подобный пример встречается в той же программе поиска максимума). Вложение допускают и альтернативы — таким способом легко заменять оператор выбора. Это показано на примере заключительного фрагмента программы «Таблица умножения» (рис. 2.13 на с. 80).

Если алгоритм удается выразить с помощью перечисленных структурных элементов, то он считается удовлетворяющим принципам структурного программирования. Сделать это бывает непросто, но вполне возможно. Доказано, что для описания любого алгоритма достаточно комбинации всего трех структурных элементов — функционального, альтернативы и цикла «пока». Цикл «пока» превращается в цикл «до», если при первом входе в него искусственно создать невыполнение условия, записанного в его начале. Цикл с параметром также несложно заменить циклом «пока», проверяя, не превысил ли параметр цикла заранее заданное значение.

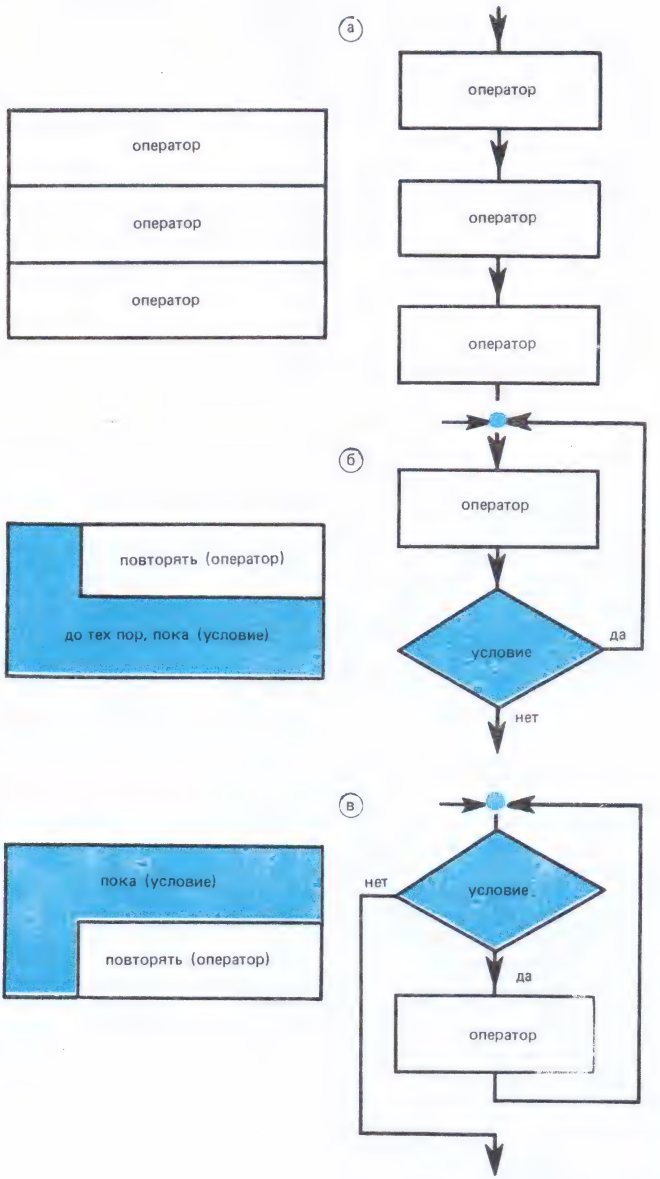


Рис. 2. 12. (начало)

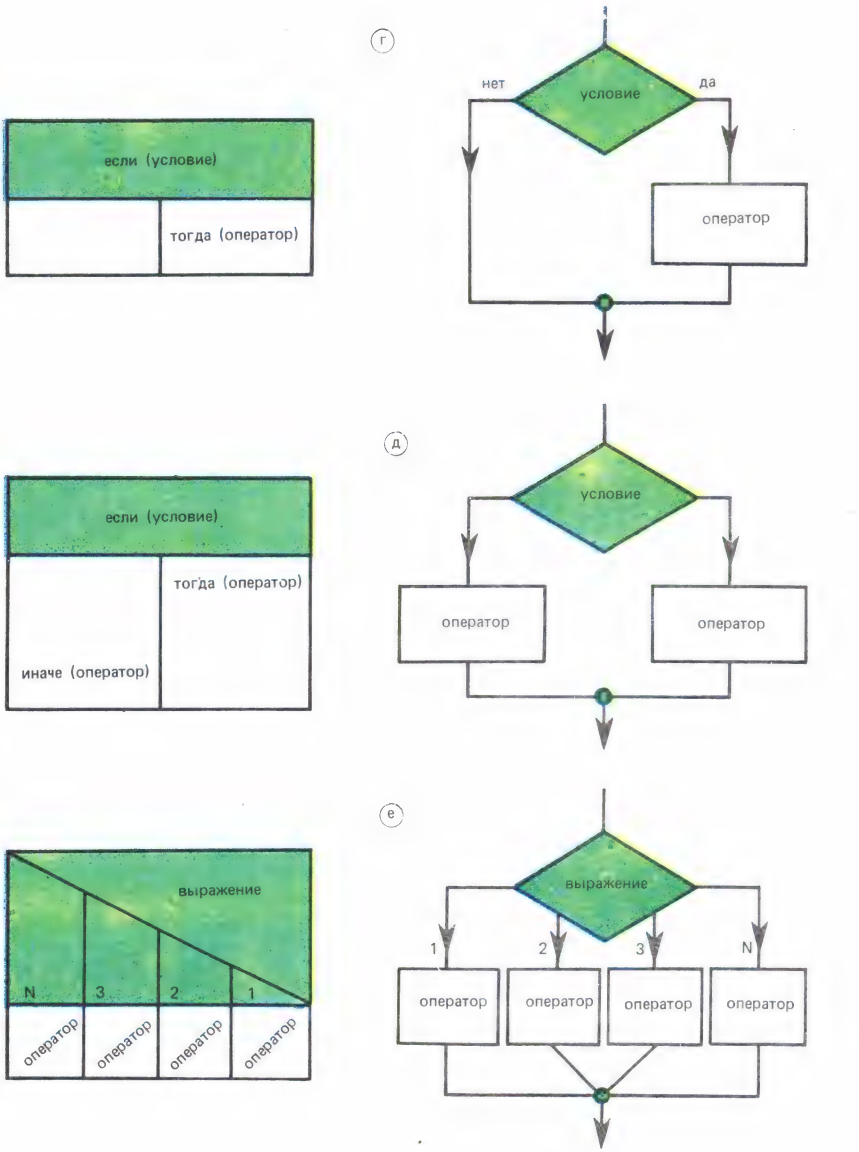


Рис. 2.12. Управляющие конструкции: *а* — следование; *б* — цикл «до»; *в* — цикл «пока»; *г* — условный оператор; *д* — альтернатива; *е* — выбор

```
110 NEXT I:
    S=1+INT(10*S/N):PRINT "ВАША ОЦЕНКА=";
120 IF S<6 THEN PRINT 2:STOP
130 IF S>5 AND S<9 THEN PRINT 3:STOP
140 IF S=9 OR S=10 THEN PRINT 4:STOP
150 IF S=11 THEN PRINT 5
```

Рис. 2.13. Замена оператора выбора с помощью вложенных альтернатив

Если же не вдаваться в такие тонкости, допустимо выразить принципы структурного программирования кратким требованием: всякий алгоритм должен состоять из таких управляющих конструкций: *следование*, *повторение* (т. е. цикл «до», цикл «пока», цикл с параметром), *выбор* (в том числе условный оператор и альтернатива).

В Бейсике эти требования не выполнимы в полной мере: здесь для реализации альтернативы и цикла «пока» приходится использовать оператор GOTO — оператор безусловного перехода на метку. В наиболее совершенных языках программирования — Паскале, Аде и т. д. — есть управляющие конструкции всех перечисленных видов.

В последних версиях Бейсика появились новые варианты управляющих конструкций, не требующих опоры на метку:

для альтернативы:

IF (условие) THEN (операторы) ELSE (операторы) IFEND

Здесь, если условие соблюдено, то выполняются операторы, записанные между словами THEN и ELSE; если же не соблюдено, то операторы, записанные между словами ELSE и IFEND.

Такая конструкция встретится нам далее в программе «Животные» (см. рис. 3.1) в строке 80;

для цикла «пока»:

WHILE (условие) (операторы) WEND

Здесь операторы, записанные перед словом WEND, выполняются, пока соблюдается условие, записанное после слова WHILE (пока). Используя эту конструкцию, строки 50 и 160 в программе «Максимум» (см. рис. 2.10) можно заменить такими:

```
50 WHILE D >= DMIN
160 D = D/2 : WEND
```

Выполнение программы, где традиционные конструкции Бейсика заменены такими структурными элементами, существенно ускоряется, так как при этом не нужно искать метки.

Подпрограммы

В программе поиска максимума многомерной функции методом «тустеп» (см. рис. 2.10) нам впервые встретилась *подпрограмма*. До этого мы обходились без подпрограмм. Между тем они представляют собой очень удобное средство программирования.

Прием их использования можно сравнить с тем, как в текстах песен пишется припев — обычно всего один раз, хотя исполняется он неоднократно. В тексте же после каждого куплета стоит только ссылка: «Припев». Встретив эту ссылку, певец переходит к тексту припева, поет его, а потом возвращается к куплету, перед которым он сделал такой переход.

В виде подпрограмм, как правило, записываются алгоритмы, вычисления по которым приходится выполнять в разных участках программы. За счет этого экономится время программиста при написании программы и память компьютера, отведенная под ее хранение. Кроме того, в программе с подпрограммами легче разбираться. Поэтому в подпрограммы нередко выносятся отдельные участки программы, даже если они встречаются в ней только один раз.

Оператор вызова подпрограммы GOSUB подобен оператору перехода GOTO в том, что он тоже нарушает ту последовательность выполнения строк, в которой они располагаются в программе. Но в отличие от оператора GOTO оператор GOSUB, передавая управление строке с указанным номером, запоминает место, где был сделан этот переход, чтобы потом, встретив команду RETURN, вернуться на это место.

Программы для ЭВМ сложнее текстов песен. Вряд ли кто припомнит песню, где припев содержал бы новый припев. Программистам же такое встречается — в подпрограммах тоже бывают повторяющиеся участки, которые целесообразно выделить в новые подпрограммы — подпрограммы второго уровня. Из этих подпрограмм также можно выделить новые подпрограммы и т. д. Сколько уровней подпрограмм допускает конкретная машина, зависит от версии Бейсика. В последних вариантах языка число уровней подпрограмм определяется объемом памяти, где должны храниться адреса переходов. Руководствуясь ими, машина сможет выбраться назад из дебрей подпрограмм в основную программу.

Предлагаем читателям испытать свой компьютер запуском несложной программы:

```
1 GOSUB 1
```

Ее выполнение через некоторое время прервется с сообщением об ошибке: «Слишком много уровней подпрограмм» или «Память компьютера исчерпана».

Место для подпрограмм

Программисты, работающие с Бейсиком, вольны писать подпрограммы в любом месте основной программы — в ее начале, в конце; некоторые ухитряются вклинивать их даже в середину программы. Опыт подсказывает, однако, два простых правила, касающихся размещения подпрограмм:

если подпрограмма имеет вспомогательный характер и не меняется от задачи к задаче, то ее помещают в конец;

если же в подпрограмме заключен существенный фрагмент расчета, для каждой задачи свой, как, например, при определении максимума многомерной функции, то ее лучше помещать в голову программ, где ее легко отыскать для замены при расчете по новой функции.

Есть еще один довод в пользу записи подпрограмм в начале программы: там машина быстрее отыскивает их при обращении к ним.

Такими же соображениями можно руководствоваться, выбирая в программе место для списков данных (операторов DATA) и описаний нестандартных функций (операторов DEF).

Этюд 12. МУЗЫКА ЦВЕТНЫХ ТОЧЕК

Во многих популярных книгах по математике описана интересная игра — «Жизнь». На поле, поделенном на клетки, располагается колония микроорганизмов. Каждая особь занимает одну клетку и имеет не более восьми соседей. Существование колонии подчиняется двум простым законам:

1) в пустой клетке возникает новая жизнь, если в окружающих ее соседних клетках имеются ровно три особи;

2) особь переходит в следующее поколение, если в окружающих клетках находится два или три соседа. При меньшем (один или ни одного) количества соседей особь погибает от одиночества, при большем (четыре и более) — от перенаселения.

Вся последующая жизнь колонии зависит от ее начальной конфигурации. Если сначала в ней было, например, три особи и располагались они цепочкой, то колония просуществует лишь два поколения и исчезнет (рис. 2.14, а). А колония из четырех особей, чьи клетки сомкнуты в

Рис. 2.14. Эволюция колонии микроорганизмов: а — содержащей три особи; б — содержащей четыре особи

квадрат, проживет без всяких изменений неограниченное время (рис. 2.14, б).

Мой приятель составил программу такой игры для своего компьютера, а затем дополнил ее расчетом возраста каждой особи. Так как дисплей у его компьютера цветной, вся колония выглядит как мозаика цветных пятен: особь, прожившая лишь одно поколение, обозначается зеленой точкой, два — желтой, три — красной, четыре — коричневой, пять и более поколений — черной.

Ваш ход в этой игре заключается только в том, чтобы ввести в память компьютера информацию о начальной конфигурации колонии. Вы запускаете программу, нажимаете клавиши, задавая координаты особей, и несколько зеленых точек загорается на дисплее. Наконец вы передаете ход машине.

Вот тут и начинается. На дисплее разыгрывается настоящая цветомузыкальная мелодия (рис. 2.15). В четком ритме одна конфигурация точек сменяется другой, цвет их тоже меняется, словно у листвы деревьев осенью: зелень блекнет и превращается в желтизну, желтизна вспыхивает багрянцем, а потом угасает до черноты. Картина на дисплее то ширится до самых его краев, то сжимается и исчезает вовсе, а то застывает в неподвижности, словно россыпь мертвых черных камней.

И каждая такая мелодия неповторима, если только вы не устанете изобретать все новые и новые начальные конфигурации.

Об усталости, впрочем, нет и речи. Времени не жалко — вы охвачены азартом, вам хочется испытать все возможные варианты исходного состояния колоний. Что будет с нею через поколение, два, три? Как отразятся на этом развитии те или иные изменения в начальной конфигурации?

Как-то раз я пришел к своему приятелю с сыном. Парень играл с машиной часа три подряд. А потом потребовал, чтобы ему показали программу игры (рис. 2.16). Мне кажется, что тот разбор дал ему больше, чем год обучения информатике по школьной программе. Он дотошно вчитывался в каждую строчку, задавал вопрос за вопросом.

Строка 10. Что такое X2 и Y2, X3 и Y3? Максимальные размеры колонии по горизонтали и по вертикали соответственно 40 и 25 клеток. А почему при вводе двумерных массивов A и B в скобках указаны числа на единицу меньше? Потому что в этой машине отсчет индексов массива ведется от нуля.

Строка 20 по своей структуре была ему вполне понятна: цикл, ввод информации о занятых клетках, всего их M. Если элемент A (X,Y) не равен нулю, значит, клетка с координатами X, Y занята. Другими словами, массив A (X, Y) описывает конфигурацию колонии. Вот только, что такое GOSUB? Переход на подпрограмму, записанную в строках 210 — 250. Участвующие там величины X3 и Y3 при включении машины автоматически становятся

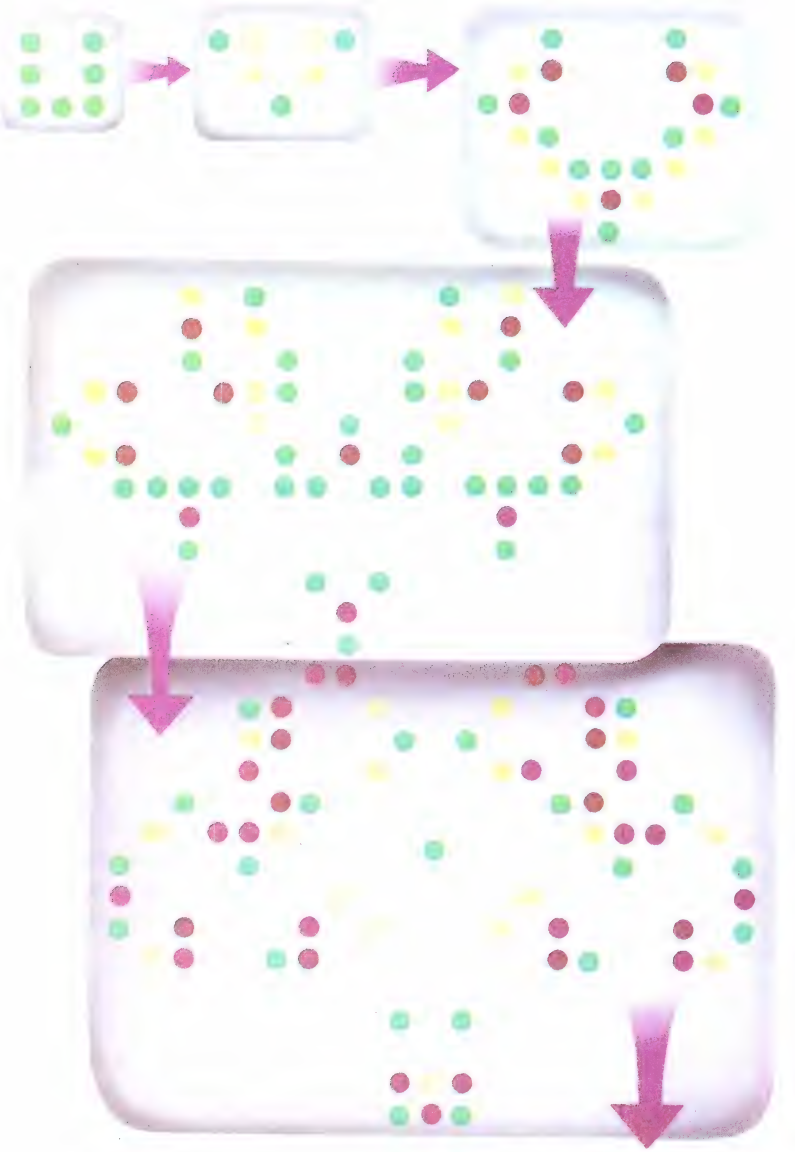


Рис. 2. 15 (начало)

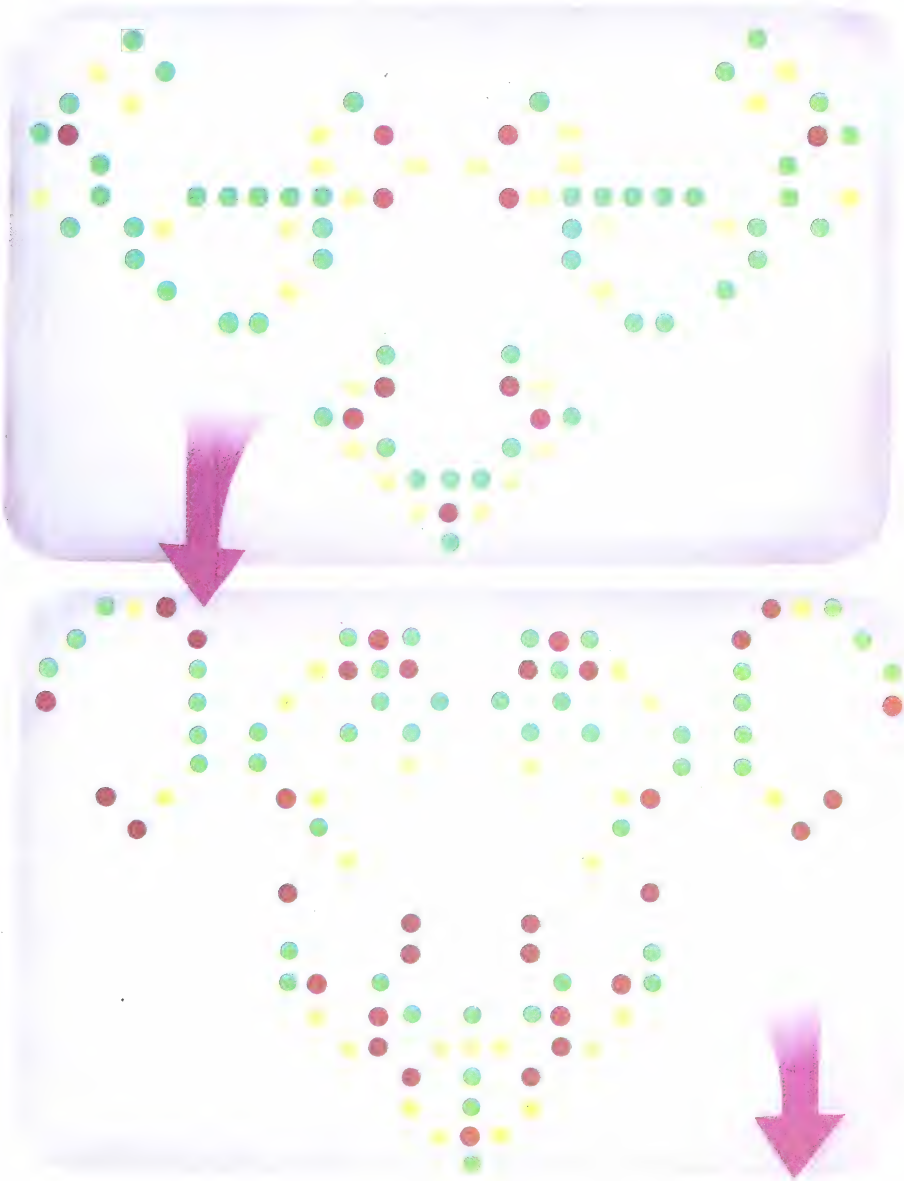


Рис. 2.15. Эволюция колонии микроорганизмов, содержащей в начальный момент семь особей

```
10 DIM A(39,24),B(39,24):X2=40:X3=0:Y2=25:Y3=0:P=0:
   INPUT"НАЧАЛЬНОЕ ЧИСЛО ОСОБЕЙ ?";M
20 FOR I=1 TO M:INPUT X,Y:A(X,Y)=1:GOSUB 210:NEXT I
30 PRINT CHR$(03):REM ОЧИСТКА ЭКРАНА
40 P=P+1:PRINT CHR$(01);P;M:M=0:REM КУРСОР В УГЛУ
50 XI=X2-1:XM=X3+1:YI=Y2-1:YM=Y3+1
60 FOR Y=YI TO YM:FOR X=XI TO XM
70 A=A(X,Y):B=0:K=0:REM ОБНУЛЕНИЕ СЧЕТЧИКА СОСЕДЕЙ:
   FOR X1=-1 TO 1:FOR Y1=-1 TO 1
80 IF X1=0 AND Y1=0 THEN 100
90 IF A(X+X1,Y+Y1)>0 THEN K=K+1
100 NEXT Y1:NEXT X1
110 CURSOR X,Y:REM УСТАНОВКА КУРСОРА
120 IF A=0 THEN 150
130 PRINT USING"#",A:
    IF K=2 OR K=3 THEN
       M=M+1:B=A+1:GOTO 160
140 GOTO 160
150                          PRINT " ":
                             IF K=3 THEN
                                GOSUB 210:M=M+1:B=1
160 IF B>9 THEN B=9:REM ИСКЛЮЧЕНИЕ ДВУЗНАЧНЫХ ЧИСЕЛ
165 B(X,Y)=B
170 NEXT X:NEXT Y
180 MAT A=B:REM ДУБЛИРОВАНИЕ МАТРИЦЫ
190 IF M>0 GOTO 40
200 END
210 IF X>X3 THEN X3=X
220 IF X<X2 THEN X2=X
230 IF Y>Y3 THEN Y3=Y
240 IF Y<Y2 THEN Y2=Y
250 RETURN
```

Рис. 2.16. Программа «Жизнь»

равными нулю, как и вообще все переменные, упоминаемые в программе. При первом же обращении к подпрограмме X2 и X3 становятся равными X, т. е. абсциссе первой введенной особи. Y2 и Y3 — равными Y, ординате первой введенной особи. При введении координат новых особей величина X2 становится равной минимальной, X3 — максимальной из их абсцисс, Y2 — минимальной, Y3 — максимальной из их ординат. С появлением новых особей за пределами поля, ограниченного величинами X2 и X3, Y2 и Y3, подпрограмма расширяет это поле соответствующим образом. Такой прием заметно ускоряет расчет новой конфигурации колонии: просматривается не весь дисплей, а только его часть, занятая особями предыдущего поколения.

А что это за оператор PRINT CHR ⊗ (03) в строке 30? Он очищает дисплей. Переменная P — номер поколения. В строке 40 он увеличивается на единицу, оператором PRINT CHR ⊗ (01) курсор переводится в левое верхнее положение, здесь на дисплей выводится номер поколения и число особей в нем. Затем рассчитывается конфигурация колонии в новом

поколении. В конце расчета будет также определено и число особей в нем, а для этого заблаговременно, в начале расчета переменная M полагается равной нулю. Новые особи могут расширить поле, занимаемое предыдущим поколением. Поэтому поле, просматриваемое при расчете (от X1 до XM и от Y1 до YM), шире, чем занимаемое в момент расчета, причем шире ровно на одну клетку во все стороны. «Вижу, — прервал пояснения сын, — X1 = X2 — 1, дальше аналогично».

Строка 60. По дважды повторенному ключевому слову FOR сын легко узнал начало двух вложенных друг в друга циклов.

Строки 70 — 170. Сын бегло просмотрел их: «M — число особей в колонии. А что такое K? Число особей в клетках, соседствующих с просматриваемой. А массив B? Он содержит информацию о состоянии колонии в будущем поколении». Сын снова углубился в этот фрагмент, мысленно проигрывая все возможности, и убедился, что он выполняет в точности те действия, которые описываются правилами игры*.

Строка 180. Что это за оператор MAT? Его название образовано сокращением слова «матрица». Так часто называют двумерный массив. Оператор MAT A = B присваивает элементам двумерного массива A(X, Y) значения соответствующих элементов двумерного массива B (X, Y). Отныне массив A (X, Y) хранит информацию о новом поколении. Управление передается на строку 40, и все повторяется вновь: на дисплей выводится вся колония в ее новом состоянии, вновь просматривается каждая клетка — выясняется, какой будет колония в следующем поколении.

— В общем, программа несложная, — заключил сын.

— Но зато какой богатой и интересной получается игра! — подумал я, глядя, как он вновь задает исходную расстановку точек.

Коды символов

Алфавит любого алгоритмического языка состоит из отдельных символов — букв, цифр, знаков препинания и некоторых специальных знаков. Все они употребляются при записи программ. Однако на клавиатуре компьютера можно видеть и такие символы, которые в программах не встречаются. Свыкнуться с их существованием поможет аналогия с пишущей машинкой. Нажатие почти каждой клавиши машинки вызывает появление того или иного знака на заправленной в нее бумаге. Но вот, допустим, клавиша, позволяющая сдвинуть каретку: после ее нажатия на бумаге ничего не возникает.

У компьютера подобных клавиш намного больше. Среди вызываемых с их помощью действий есть, например, такие (о цифрах в скобках будет сказано позже):

перевод курсора в левый верхний угол дисплея (21);

то же, с очисткой дисплея (22);

---

* Чтобы приведенная программа была выполнима на компьютерах с черно-белым дисплеем, возраст особи в ней выражается не цветом, а цифрой — номером поколения.

перевод курсора на одну из соседних позиций, например вправо (19) или влево (20).

Можно, конечно, изобрести особые символы для обозначения каждого из таких действий, ввести их в алфавит языка, узаконив их употребление в программах. Но не слишком ли усложнится тогда алфавит? Не станет ли он трудным для запоминания?

Было решено пойти по другому пути. Все знаки из алфавита языка закодированы числами от 0 до 255. Так же закодированы различные элементарные действия, выполняемые на компьютере, например перечисленные выше манипуляции с курсором. Числа, указанные в скобках, и представляют собой эти коды. Так же, в скобках, они проставляются и в операторах, записываемых в программе для совершения таких действий. Например: PRINT CHR ⊗ (21).

Матричные операторы

Наиболее совершенные версии языка Бейсик имеют целый ряд специальных операторов для обработки одномерных и двумерных массивов. Процесс программирования в этом случае существенно облегчается.

Например, в программе «Жизнь» нужно было перенести значения элементов матрицы B в матрицу A. Не будь матричных операторов, для этого нужно было бы записать такую конструкцию:

```
FOR X = 0 TO 39 : FOR Y = 0 TO 24 : A (X, Y) = B (X, Y) : NEXT Y : NEXT X.
```

В приведенной программе (см. рис. 2.16) это выглядит намного проще:

```
MAT A = B
```

В программе «Кассир кинотеатра» (см. рис. 2.8) план зрительного зала тоже мог бы «нарисовать» *матричный оператор*:

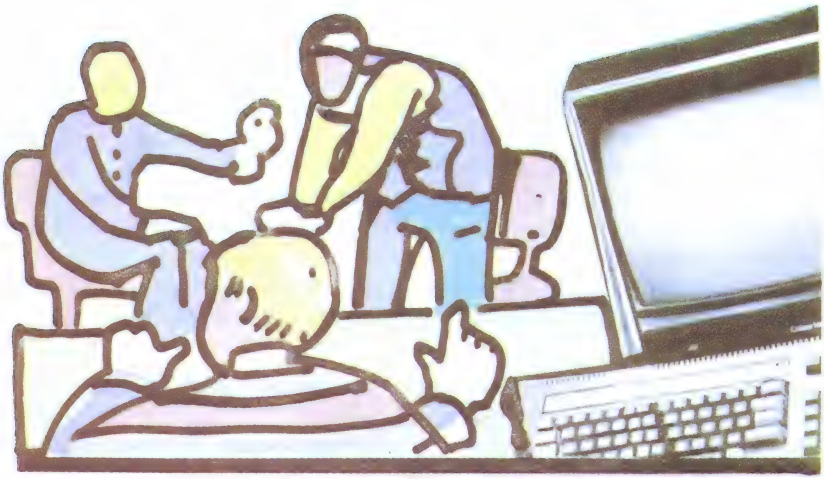
```
MAT PRINT A
```

Одним коротким оператором:

```
P = DET MAT A
```

можно вычислить определитель квадратной матрицы A и присвоить это значение переменной P. Располагая таким оператором, несложно написать короткую программу решения системы линейных алгебраических уравнений методом Крамера.

Наконец, матричные операторы существенно ускоряют счет по программе.



Этюд 13. КОМПЬЮТЕР УЧИТСЯ НА СВОИХ ОШИБКАХ

Лучший способ освоить компьютер — это игры. Нужно только найти хорошего ведущего. Моему сыну повезло. С тех пор как приятель познакомил его с игрой «Жизнь», он жить не может без компьютерных развлечений. Хлебом не корми, а покажи ему какую-нибудь новую игру. Наигравшись, просит объяснить ее программу. Таким объяснениям нет цены! Каждое — это отличный урок программирования.

Мы условились с приятелем, что он станет подбирать игры так, чтобы за разбором их программ сын осваивал все более сложные темы. Компьютерная игротека превратилась в школу программирования. Урок в ней шел за уроком, и вот настала очередь изучать литерные переменные и массивы.

— На сегодня я приготовил игру «Животные, — объявил мой приятель, когда мы снова пришли к нему в гости. Компьютер был уже включен, и сын устроился за дисплеем, ожидая объяснений.

— Условие игры единственное, — продолжал мой друг, — честно отвечать «да» или «нет» на вопросы машины. Задумываешь животное. Машина спрашивает, например, «Может ли оно ползать?». Допустим, ты отвечаешь «да». Машина тут же выдает свою догадку: «Это змея? Угадала?». Если ты отвечаешь «да» — игра закончена, «нет» — машина сдается, если у нее в запасе нет нового уточняющего вопроса. В этом случае она попросит тебя

назвать задуманное. Пусть это будет крокодил. После этого машина проверяет, не было ли крокодила в списке ее более ранних предположений. «Да» — игра прерывается из-за твоей нечестности. «Нет» — ЭВМ учится на своей ошибке. Она просит сформулировать и набрать на клавиатуре альтернативный вопрос, ответив на который, можно уловить разницу между задуманным (крокодил) и последним предположением машины (змея). Эти новые сведения машина запомнит. В следующий раз на крокодиле ее не проведешь. Ну как, понятно я объясняю?

— Да. Только. . . что такое «альтернативный вопрос»?

— Вопрос, на который можно ответить только «да» или «нет». Так что — сыграем?

Об этом можно было не спрашивать. «Я уже задумал!» — выпалил сын и нажал клавишу RUN, потом ВК. Тотчас на дисплее появилось:

ЭТО КОТ? УГАДАЛА (1 — ДА, 0 — НЕТ)?

Сын, торжествующе улыбаясь, нажал клавишу 0.

— СДАЮСЬ, КТО ЭТО? — сразу запросил пощады незадачливый компьютер.

— Кит! — воскликнул сын, довольный первой победой, и набрал это слово. Компьютер торопливо выписывал на дисплее:

ЗАДАЙТЕ АЛЬТЕРНАТИВНЫЙ ВОПРОС,
ОТВЕТИВ НА КОТОРЫЙ МОЖНО ОТЛИЧИТЬ КИТ ОТ КОТ?

— Он пишет с ошибками! — недоуменно обернулся сын к приятелю.

— Не обращай внимания, — успокоил его тот. — Потом я тебе все объясню. Набирай то, о чем он тебя просит. Можешь без вопросительного знака.

Сын набрал:

ЖИВЕТ ОНО В ВОДЕ

На дисплее загорелось:

ПРАВИЛЬНЫЙ ОТВЕТ (1 — ДА, 0 — НЕТ)?

— Опять непонятно! — недоумевал сын.

— Ну, а как бы ты сам ответил на свой вопрос? Да, оно живет в воде, не так ли? Правильный ответ — «да». Выбор в таких случаях может определяться и отрицательным ответом: нет, оно не живет в воде. Две возможности ответа — богаче и интереснее игра. Без этого она сильно усложнилась бы, уверяю тебя. Вводи единицу.

Сын ввел. Машина спрашивала дальше:

БУДЕТЕ ЕЩЕ ИГРАТЬ?

— Задумай новое животное, — подсказал приятель, и сын нажал клавишу 1. Игра продолжалась:

```
ЭТО КОТ? УГАДАЛА (1 — ДА, 0 — НЕТ)? 0
ЖИВЕТ ОНО В ВОДЕ? 1
ЭТО КИТ? УГАДАЛА (1 — ДА, 0 — НЕТ)? 0
СДАЮСЬ, КТО ЭТО? РУСАЛКА.
```

Протокол игры, строка за строкой, заполнял верхнюю половину дисплея.

— Русалка? Хвалю ваше остроумие! — приятель похлопал моего сына по плечу. — Может быть, начнем разбор программы?

Сын протянул руку к клавише LIST.

— Не спеши! — остановил его приятель. — Сначала давай разберем грамматические ошибки машины. Ты уже понял, в чем их причина? Нет? Тогда реши мне одну простенькую задачку. Представь, что тебе время от времени нужно подсчитывать сумму четырех чисел A + X + B + Y. Числа A и B — постоянные, равные, допустим, 100 и 3; X и Y — величины переменные, перед каждым подсчетом ты вводишь их сам. Какую программу ты предложишь для решения такой задачи?

Сын, не долго думая, написал:

```
10 A = 100
20 B = 3
30 INPUT «X, Y?»; X, Y
40 C = A + X + B + Y
50 PRINT C
```

— Совершенно верно! — одобрил написанное приятель. Теперь слушай меня внимательно. Точно так же, как числа мы записываем цепочками цифр, мы можем записать в запоминающем устройстве машины цепочку букв и использовать ее в определенных операциях. Запишем, например: «ЗАДАЙТЕ АЛЬТЕРНАТИВНЫЙ ВОПРОС, ОТВЕТИВ НА КОТОРЫЙ МОЖНО ОТЛИЧИТЬ». И еще: «ОТ». По ходу игры ты вводил в машину свои цепочки: КИТ, РУСАЛКА, . . . Выстраивая такие буквенные цепочки одну за другой, машина составляет свои фразы:

```
ЗАДАЙТЕ АЛЬТЕРНАТИВНЫЙ ВОПРОС, ОТВЕТИВ НА КОТОРЫЙ МОЖНО ОТЛИЧИТЬ КИТ ОТ КОТ
ЗАДАЙТЕ АЛЬТЕРНАТИВНЫЙ ВОПРОС, ОТВЕТИВ НА КОТОРЫЙ МОЖНО ОТЛИЧИТЬ РУСАЛКА ОТ КИТ
```

По той программе, которую я ввел в компьютер, он не занимается никаким грамматическим согласованием частей таких фраз. Он выполняет самую простую операцию — составляет заданные цепочки букв в единую строчку. Ты спросишь: как же записать такую операцию в программу? Для этого прежде всего каждой из складываемых цепочек надо присвоить свое имя. Причем оно должно чем-то отличаться от имен, под которыми мы понимаем числа. Таким отличительным признаком в Бейсике служит добавочный символ:

$

Программисты называют его «солнышком»*. И если ты встречаешь в программе обозначение A ⊗, знай — за ним скрывается некая строчка символов, в частности букв. Ты, вижу, хочешь что-то спросить?

— Зазор между словами — тоже символ?

— Зазор? Программисты называют его *пробелом*. Да, пробел — это полноправный символ. Без него не обойтись. Без него слова во фразах «слипались» бы: ЗАДАЙТЕАЛЬТЕРНАТИВНЫЙВОПРОС. . . А благодаря ему получаются вполне понятные фразы и словосочетания. Положим, например:

A ⊗ = «ЗАДАЙТЕ АЛЬТЕРНАТИВНЫЙ ВОПРОС, ОТВЕТИВ НА КОТОРЫЙ МОЖНО ОТЛИЧИТЬ»
B ⊗ = «ОТ»
X ⊗ = «РУСАЛКА»
Y ⊗ = «КИТ»

Теперь запомни, как называется то, с чем мы только что познакомились. A ⊗ и B ⊗ — это *литерные константы*. X ⊗ и Y ⊗ — *литерные переменные*. Слово «литерный» указывает, что характер таких переменных и констант — не числовой, они представляют собой цепочки произвольных символов. Литерные константы — цепочки неизменные. Литерные переменные — строчки, чьё содержание может меняться по ходу выполнения программы. Вот и в той программе, по которой машина составляет свои «неграмотные» фразы, цепочки A ⊗ и B ⊗ используются в неизменном виде, а цепочки X ⊗ и Y ⊗ ты задаешь по своему усмотрению.

В этой программе каждая фраза конструируется так: A ⊗ + + X ⊗ + B ⊗ + Y ⊗. Плюс означает здесь сложение буквенных цепочек, их выстраивание одна за другой. Всю фразу можно обозначить особым символом, например, C ⊗ и написать: C ⊗ = A ⊗ + X ⊗ + B ⊗ + Y ⊗. Здесь знак равенства означает операцию присваивания, смысл которой тот же, что и для числовых величин. Ты парень сообразительный и, наверное, сумеешь составить программу, по которой машина сможет придумать сколько угодно подобных фраз.

— Да мы ведь ее уже составили! — сын направил взгляд на программу для подсчета суммы чисел.

— Молодец! — похвалил его мой друг, глядя, как сын написал свою новую программу:

```
10 A ⊗ = «ЗАДАЙТЕ АЛЬТЕРНАТИВНЫЙ ВОПРОС, ОТВЕТИВ НА КОТОРЫЙ
       МОЖНО ОТЛИЧИТЬ»
20 B ⊗ = «ОТ»
30 INPUT «X ⊗, Y ⊗?»; X ⊗, Y ⊗
40 C ⊗ = A ⊗ + X ⊗ + B ⊗ + Y ⊗
50 PRINT C ⊗
```

* В типографском наборе такой знак отсутствует, поэтому в книге он заменен символом ⊗

— Молодец! — повторил он. — Все верно: ввод, сложение, присваивание, вывод... Есть еще много других операций над строками. В дальнейшем мы познакомимся с ними и с их особенностями. Например, в программе литерные константы записывают, ограждая кавычками, но в операциях над строками кавычки отбрасываются. Так, с помощью кавычек можно записать и пробел, и длинную пустую строку. Но все это частности, которые понадобятся нам не скоро. Вернемся к нашей игре. Мне кажется, теперь ты уже сможешь разобраться в ее программе. Почти все необходимые для этого понятия мы с тобой уже освоили. Осталось только узнать, что такое *литерный массив*. Это массив, элементы которого — литерные переменные. Вводится он с помощью того же оператора DIM, что и числовые массивы. Например:

10 DIM B ⊗ (200) 20

Такая запись означает, что в литерном массиве B ⊗ (200) может насчитываться до 200 элементов, а в каждом элементе содержится 20 знаков. Если за скобками никакое число не проставлено, максимальная длина элемента полагается равной 16 символам по умолчанию. Да, — прибавил друг, подумав, — еще одно пояснение.

— Действия машины в игре «Животные» подобны лазанью по дереву (см. рисунок на 3-й стороне обложки), каждая ветвь которого при ответе человека раздваивается: «да» — налево, «нет» — направо. На ветвях «висят» догадки машины, названия различных животных. Мы к этой схеме еще вернемся. А вот и сама программа, — он нажал клавишу LIST, и под протоколом игры появились занумерованные строки (рис. 3.1).

```
10 DIM B¤(2047),O¤(2047),P¤(2047)
20 I=0:O¤(I)="КОТ":GOTO 70
30 O¤=O¤(I):IF I=0 THEN I=1:GOTO 50
40 I=2*I:IF A¤=P¤(I) THEN I=I+1
50 IF B¤(I)="" THEN 90:REM МАШИНА СДАЕТСЯ
60 PRINT B¤(I);:INPUT A¤:IF A¤<>P¤(I) GOTO 30
70 PRINT "ЭТО - "O¤(I);:INPUT "? УГАДАЛА
   (1-ДА,0-НЕТ)?";P:IF P=0 GOTO 30
80 INPUT "БУДЕТЕ ЕЩЕ ИГРАТЬ?";P:IF P=1 THEN 20
   ELSE STOP
90 INPUT "СДАЮСЬ. КТО ЭТО?";O¤(I)
100 FOR J=0 TO I-1:IF O¤(J)=O¤(I) THEN PRINT
    "НЕЧЕСТНО ИГРАЕТЕ.":GOTO 80
110 NEXT J:REM КОНЕЦ ЦИКЛА ПРОВЕРКИ
120 PRINT "ЗАДАЙТЕ АЛЬТЕРНАТИВНЫЙ ВОПРОС,
    ОТВЕТИВ НА КОТОРЫЙ МОЖНО ОТЛИЧИТЬ "O¤(I)
    " ОТ "O¤;:INPUT B¤(I):INPUT"ПРАВИЛЬНЫЙ
    ОТВЕТ(1-ДА,0-НЕТ)?";P¤(I):GOTO 80
```

Рис. 3.1. Программа «Животные 1»

— Прочти еще раз внимательно протокол игры. Ты видишь, что машина начинает ее, следуя по строкам 20, 70, . . . Далее, если первая догадка машины «это кот?» неверна, управление передается на строку 30. Здесь переменная О ⊗ дублирует элемент О ⊗ (0). Далее программа ведет нас по строкам 50, 90, так как элемент В ⊗ (0) еще не принял никакого значения и равен пробелу, с которым он сравнивается в строке 50. В строке 90 ответ игрока помещается на первую ветку дерева игры. Далее выполняется строка 100, строка 110 с проверкой, честно ли ведет себя игрок; потом — строки 120, 80 и, если игрок продолжает игру, — строка 20. Ты решил продолжать. Начальный путь машины в новом туре игры вновь пролегает по строкам 70, 30, 50. Но тут, поскольку элемент В ⊗ (I) уже не пуст, машина идет в новом направлении, на строку 60. Здесь она задает свой первый альтернативный вопрос. Допустим, твой ответ А ⊗ совпал с занесенным в память машины верным ответом Р ⊗ (I). Тогда ЭВМ идет на строку 70. Не совпал — она переходит на строку 30. Рассмотрим первый из этих двух вариантов. Он возникает, когда элемент массива не пуст.

Очутившись на строке 70, машина выводит на дисплей свою догадку и, если она верна, ЭВМ спрашивает игрока о желании играть дальше. Если же не угадала, переходит на строку 30, дублирует тут свою неверную догадку О ⊗ (I), идет далее на строку 40, так как I ≠ 0, здесь удваивает индекс I, т. е. перемещается на новый ярус «дерева ответов». Посмотри на него, проследи нумерацию ветвей: когда какая-то ветвь раздваивается на два побега, то левый побег получает номер, вдвое больший, чем сама ветвь, а правый побег — номер, на единицу больший, чем левый. В зависимости от совпадения или несовпадения А ⊗ и Р ⊗ (I) машина «снимает» свою очередную догадку либо с левого, либо с правого побега. О каждой вновь рассматриваемой ветви машина осведомляется, не пуста ли она — взгляни еще раз на строку 50. Если пуста, ЭВМ сдается и просит игрока сформулировать альтернативный вопрос, отличающий неизвестное ей животное от известных. Вот и вся программа!

Сын снова «прилип» к дисплею, а приятель подсел ко мне на диван:

— Ну, как игра?

— Отличная! Я даже придумал, как ее усовершенствовать. Скоро 1 апреля — так я проведу с ее помощью юмористическо-социологическое обследование нашего отдела. Будем загадывать не животных, а друг друга: «Задайте альтернативный вопрос, ответив на который можно отличить прогульщика Муркина от сверхпунктуального бездельника Чуркина». . .

Я снова посмотрел на сына:

— Понравилась ему твоя игра. Наверное, в следующий раз он попросит сыграть в нее снова. Так что же — тогда опять придется

вводить в машину все прежние вопросы и ответы? Это ведь утомительно! Есть у твоего компьютера дисковод?

— Есть, — успокоил меня друг. — О нем я собираюсь поговорить в следующий раз. Когда вы уйдете, я перепишу на него дерево игры. Это будет сюрприз для твоего сына, когда он снова захочет поиграть в эту игру.

Обозначение литерных переменных

Переменную, хранящую цепочку символов, программисты называют по-разному: литерная, строковая, текстовая, символьная, стринговая (от английского string — веревка, вереница). По-разному именуются и операции над такими переменными. Их можно, например, объединять друг с другом: «ТРАК» + «ТОР» = «ТРАКТОР». Такая операция называется и сложением, и склеиванием, и конкатенацией. Знак ⊗, отличающий литерную переменную, одни программисты называют *солнышком*, другие — *клопом*. Есть у него и «официальное» название — знак денежной единицы. На импортных машинах его синоним — знак доллара $.

Как еще можно использовать программу!

Игра «Животные», если ее вести на иностранном языке, может стать эффективным средством овладения иноязычной лексикой. При этом в разных турах игры можно загадывать не любые предметы и явления, а только те, какие относятся к изучаемым темам: «Моя комната», «Летний отдых», «Моя семья» и т. д. ЭВМ должна отгадывать предметы, входящие в лексический минимум по данной теме.

Литерные массивы в этом случае нужно сделать двумерными: второй индекс каждого элемента будет соответствовать номеру языка: 1 — русский, 2 — английский, 3 — немецкий и т. д.

После того как машина сдастся и попросит назвать задуманное, к дисплею должен подсесть полиглот для заполнения пустых элементов двумерных литерных массивов.

Играющий перед началом игры выбирает язык, а в ее процессе имеет возможность получить подсказку на родном языке, изменив второй индекс массива вопросов.

Составьте такую программу.

Этюд 14. ЕСЛИ БЫ Я РАБОТАЛ КАССИРОМ В БАНКЕ

Когда художник, взявшийся иллюстрировать мою книгу, пожелал изучить программирование, это показалось мне причудой. Но за художником — новая неожиданность! — потянулся редактор книги. Он, оказывается, уже слышал о программах, которые составляют всевозможные каталоги, редактируют тексты и даже переводят с одного языка на другой. Он стал настаивать, чтобы я рассказал об этом в книге.

Сошлись мы на том, что о подобных программах я напишу во второй части книги. А чтобы переход к ней не был слишком резким, начну ее с задачи, при решении которой пригодились бы знания, изложенные в первой, посвященной главным образом числовым выкладкам.

Несложная задача, где наряду с числовыми выкладками шла бы обработка какого-то текста. . . Удастся ли мне найти такую? Задача, где на равных выступали бы цифры и буквы. . . И тут меня осенило: ведь были же времена, когда числа записывались не цифрами, а буквами. Одним из способов такой записи мы изредка пользуемся по сей день — это римские числа. Никаких вычислений с ними мы, конечно, не производим — слишком они для этого неудобны. В таких случаях мы переводим их в привычную десятичную систему счисления, записываем арабскими цифрами.

Так я пришел к искомой задаче, точнее, сразу к двум: составить программу для перевода римских чисел в арабские и обратно.

Не буду описывать, как долго я не мог составить алгоритмы того и другого перевода, сколько сложных вариантов перебрал,

пока не напал на неожиданно простой. Все получилось, как в поговорке: «Ищу рукавицы, а они за поясом». Подобными алгоритмами мы пользуемся по многу раз в день, когда ведем денежные расчеты и прикидываем, какими банкнотами и монетами можно выдать требуемую с нас сумму денег.

Представьте, что вы — кассир в банке, что выдаваемая сумма достаточно велика и что в вашем распоряжении есть все существующие денежные знаки — от сотенной купюры до копеечной монеты. Естественно начать выдачу с самых крупных, сотенных, купюр, отсчитывая их одну за другой, пока в остатке не окажется меньше ста рублей. Продолжим ту же операцию с остатком, только на сей раз будем вести выдачу купюрами, следующими по старшинству, пятидесятирублевыми и т. д.

Аналогия с переводом арабских чисел в римские здесь полная. Десятичное число — выдаваемая сумма. Знаки, употребляемые в римской системе счисления, — купюры. Вычитайте из заданного десятичного числа те величины, которые соответствуют этим знакам, и одновременно выписывайте эти знаки один за другим, формируя искомое римское число.

В подобных задачах фигурируют довольно «неудобные» числовые ряды, члены которых не подчиняются простой алгебраической закономерности. В самом деле, попробуйте-ка выразить с помощью короткой формулы достоинства купюр: 100, 50, 25, 10, 5, 3, 1.

То же самое можно сказать и про числа, соответствующие знакам римской системы счисления. Если брать лишь отдельные употребляемые в ней буквы, все кажется не таким уж запутанным: 1000 — M, 500 — D, 100 — C, 50 — L, 10 — X, 5 — V, 1 — I. В этом ряду числа уменьшаются попеременно то в два, то в пять раз. Но весь фокус в том, что в римских числах встречаются и двухбуквенные сочетания, скажем 40 — XL. И если перечислить по порядку те и другие обозначения, то ряд получается довольно запутанным: 1000 — M, 900 — CM, 500 — D, 400 — CD, 100 — C, 90 — XC, 50 — L, 40 — XL, 10 — X, 9 — IX, 5 — V, 4 — IV, 1 — I.

К счастью, в Бейсике на подобные случаи есть оператор DATA, позволяющий задавать любые наборы данных в виде списка. Так мы и поступим с только что приведенным набором чисел и символов. И пусть они стоят в списке так же, как были названы, — парами. Их ведь и использовать удобнее всего тоже парами: вычли из переводимого десятичного числа очередную величину, стоящую в списке, и тут же приписали соответствующий знак к формируемому римскому числу.

Поместим список в первой строке программы перевода:

```
10 DATA 1000,M,900,CM,500,D,400,CD,
        100 ,C,90  ,XC,50  ,L,40  ,XL,
        10  ,X,9   ,IX,5   ,V,4   ,IV,
        1   ,I
```

В этой же строке поместим оператор, который позволит нам ввести переводимое число:

:INPUT «АРАБСКОЕ ЧИСЛО?»;N

Начинаем перевод. С помощью оператора READ считываем очередную пару. На первый раз она будет такой: 1000, M. Именно такие значения мы присвоим вначале переменным, участвующим в дальнейшей реализации нашего алгоритма. Как их лучше обозначить? Природа у них различная: первая из этих переменных — числовая, вторая — литерная, к ее имени добавляется «солнышко». Поскольку это так, можно обозначить их одной и той же буквой. Первую — A, вторую — A ⊗. «Солнышко» не даст спутать их.

20 READ A, A ⊗

Знак M, считанный этим оператором при первом его выполнении, войдет в формируемое римское число, если переводимое арабское число N больше или равно 1000. Тогда нарастим значение литерной переменной N ⊗, вычтем из N величину A и присвоим значение разности той же переменной N, чтобы описанный процесс можно было повторять циклически:

30 IF A <= N THEN N ⊗ = N ⊗ + A ⊗ : N = N − A : GOTO 30

Рано или поздно переменная N станет меньше вычитаемой из нее величины A. Тогда с помощью оператора READ считаем очередную пару из списка, заданного оператором DATA. Но прежде чем перейти к считыванию, проверим, не уменьшилось ли N до нуля по ходу вычитания:

40 IF N > 0 GOTO 20

Вот вроде бы и все... Неужели все? Неужели так просто? Я ввел программу (рис. 3.2) в компьютер, задал в качестве примера число 333. Компьютер выписал на дисплее CCCXXXIII. Задал 1024 — на дисплее появилось MXXIV. Задал 1988 — и получил в ответ MCMLXXXVIII.

```
10 DATA 1000,M,900,CM,500,D,400,CD,
        100 ,C,90 ,XC,50 ,L,40 ,XL,
        10  ,X,9  ,IX,5  ,V,4  ,IV,
        1   ,I:
   INPUT"АРАБСКОЕ ЧИСЛО?";N:N¤=""
20 READ A,A¤:REM СЧИТЫВАНИЕ ПАРЫ
30 IF A<=N THEN
             N¤=N¤+A¤:N=N-A:GOTO 30
40 IF N>0 GOTO 20
50 PRINT"РИМСКОЕ ЧИСЛО "N¤
```

Рис. 3.2. Программа «Арабские — римские»

Все верно! И я приступил ко второй программе — для перевода римских чисел в арабские.

Не долго думая, переписал две первые строки из предыдущей программы, несколько изменив их, и остановился в раздумье.

Новая программа интуитивно представлялась мне намного сложнее предыдущей. На это настраивало привычное представление о том, сколь неудобны выкладки с римскими числами. А здесь, казалось бы, без них не обойтись. Если при переводе арабского числа в римское мы шаг за шагом отнимаем от переводимого числа 1000, 900, 500 и т. д., то по ходу обратного перевода нам придется отнимать от исходного римского числа, допустим того же MCMLXXXVIII, числа M, CM, L и т. д.

Но ведь отнимать их можно простым отбрасыванием, — тотчас успокоил я себя. В Бейсике есть функции, «вычисляемые» от литерных переменных, и среди них как раз те, что мне нужны для этой цели.

К примеру, функция LEN(A ⊗) определяет, сколько знаков насчитывает литерная переменная A ⊗. Так, LEN («M») = 1, LEN («CM») = 2.

Функция MID ⊗(A ⊗, J, K) вырезает из строки A ⊗ ее часть (подстроку) длиной K, начиная с J-го знака. Вслед за обозначением этой функции стоит «солнышко»: оно напоминает, что значение функции — тоже строка.

Функция LEFT ⊗ (A ⊗, J) берет J левых знаков от строки A ⊗, функция RIGHT ⊗ (A ⊗, K) — K правых знаков.

С помощью этих функций нетрудно совершить те «вычитания», которые мне нужны. Как, например, из числа MCMLXXXVIII «вычесть» число M? Сначала, конечно, надо убедиться, что вычитаемое есть в уменьшаемом и стоит точно с левого края. На Бейсике это запишется так:

```
«M» = LEFT ⊗ («MCMLXXXVIII», LEN («M»))
```

А потом возьмем от уменьшаемого столько правых знаков, число которых определяется как длина уменьшаемого минус длина вычитаемого:

```
RIGHT ⊗ («MCMLXXXVIII», LEN(«MCMLXXXVIII») — LEN(«M»)).
```

Так на примерах я понял, какой должна быть следующая строка моей программы: проверяю, стоит ли знак A ⊗ с левого края у переводимого римского числа N ⊗; если стоит, то «вычитаю» его только что описанным способом, результат присваиваю все той же переменной N ⊗, а к переменной N прибавляю величину A. Повторяю эту операцию вновь и вновь. Если знак A ⊗ не обнаружился с левого края у переменной N ⊗, считываем следующую пару A,A ⊗ . Но перед этим проверяем, не исчерпана ли переменная N ⊗. Для этого сравниваем ее с пустым местом —

```
10 DATA 1000,M,900,CM,500,D,400,CD,
        100 ,C,90 ,XC,50 ,L,40 ,XL,
        10  ,X,9  ,IX,5  ,V,4  ,IV,
        1   ,I:
   INPUT"РИМСКОЕ ЧИСЛО?";N¤:N=0
20 READ A,A¤:REM СЧИТЫВАНИЕ ПАРЫ
30 IF A¤=LEFT¤(N¤,LEN(A¤)) THEN
   N=N+A:N¤=RIGHT¤(N¤,LEN(N¤)-LEN(A¤)):GOTO 30
40 IF N¤>"" GOTO 20
50 PRINT"АРАБСКОЕ ЧИСЛО "N
```

Рис. 3.3. Программа «Римские — арабские»

заключенным в кавычки. Если исчерпана, вывожу на дисплей число N.

Прогонка программы (рис. 3.3) с исходными данными из тех же примеров дала верные результаты.

Чтение двоичных чисел

Известен простой алгоритм «чтения» двоичных чисел. Под каждым его знаком справа налево приписываются последовательные натуральные степени двойки, начиная с нулевой: $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$ и т. д. Каждое из этих чисел умножается на стоящую над ним единицу или нуль. Полученные произведения складываются (рис. 3.4).

номер разряда числа	6	5	4	3	2	1	0
двоичное число	1	1	0	1	1	0	1
2 в степени, равной номеру разряда	64	32	16	8	4	2	1

$1 \cdot 64 + 1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 2 \cdot 1 = 109$

Рис. 3.4. Перевод двоичного числа в десятичную систему счисления

Такой алгоритм нетрудно претворить в программу на Бейсике. Для этого двоичное число, цепочку единиц и нулей, очевидно, следует рассматривать как литерную переменную, с помощью функции MID ⊗ извлекать ее последовательные знаки, начиная с правого, превращать их в числа и умножать на последовательные степени двойки.

Приведенная на предыдущих страницах программа перевода арабских чисел в римские подскажет, как запрограммировать перевод чисел из десятичной системы в двоичную. Из исходного числа следует вычитать последовательно убывающие степени двойки, следя за тем, чтобы разность каждый раз была неотрицательной. Если очередное вычитание выполнимо, нужно ставить в формируемой записи двоичного числа единицу, не выполнимо — нуль.

Синонимы Бейсика

В символах функций, преобразующих литерные переменные, различия в версиях Бейсика проявляются, по-видимому, в наибольшей степени. Вот конкретный пример. Требуется из переменной A ⊗, хранящей текст «электронно-вычислительная машина», вырезать фрагмент «вычислитель» и сделать его значением переменной B ⊗. На разных машинах это делается по-разному:

«ИСКРА 226»:	B ⊗ = STR(A⊗,12,11)
СМ-4 И ДВК-2М:	B ⊗ = SEG ⊗(A⊗,12,22)
IBM PC:	B$ = MID$(A$,12,11)

Числа в скобках указывают на место вырезаемого фрагмента в переменной A ⊗: 12 — позиция первого знака, 22 — последнего, 11 — число знаков.

Конечно, такой разнобой затрудняет перевод программ с одной версии языка на другой. Но отчаиваться не стоит. При написании программ важно прежде всего знать, что можно делать с литерными переменными, а уж о том, как ту или иную операцию реализовать, можно справиться в инструкции, приложенной к машине.

Этюд 15. КОМПЬЮТЕР ЛИКВИДИРУЕТ СВОЮ НЕГРАМОТНОСТЬ

Когда на моем домашнем столе появился персональный компьютер, когда он впервые выполнил ту программу, которую я в него ввел, я испытывал радость, ни с чем не сравнимую! Я был готов петь и плясать, показывать результаты всем знакомым!

Мне казалось, что у меня стало одним чувством больше. Нет, дело не в том, что я стал как-то по-другому, богаче и разнообразнее, воспринимать окружающий мир. Новое ощущение возникло внутри меня самого. Я казался самому себе могущественным чародеем, хитроумным магом. Да и сейчас, когда нажимаю клавиши на пульте своего компьютера, бормоча строки программы, вижу себя со стороны этаким стариком Хоттабычем с его заклинающим «трах-тибидох-тибидох . . .».

Прихожу домой вечером, усталый после работы. Дай, думаю, посижу полчасика за компьютером. Сижу . . . Вдруг слышу — за окном шум. Оказывается — утро, город ожил, народ потянулся на работу. Маги, насколько мне известно, тоже любили работать по ночам. Я их понимаю.

Я часто ловлю себя на желании любую задачу, любую даже мелкую житейскую проблему перевести на Бейсик и решить на своем компьютере.

Жена планирует на воскресенье званый обед. Придут Поповы, придут Ивановы, Копыловы, придет Саша Букин, старый холостяк. Народу будет немного, всех можно по пальцам перечесть.

Но я все равно сажусь за дисплей. Я придумываю программу, которая составит список приглашенных в алфавитном порядке. Я подвожу под свою работу серьезное обоснование: моя программа пригодилась бы, например, ученому, который, заканчивая работу над книгой или отчетом, приводит в порядок список использованной литературы — расставляет его позиции по алфавиту.

Если бы нашими гостями были не Поповы и Ивановы, а Смиты и Джонсоны, то составить такую программу было бы намного легче. Любая персональная ЭВМ знает латинский алфавит. Ее предки имели дело только с латинскими буквами, составляющими почти во всех языках программирования имена переменных и названия операторов. Работать с текстами, русскими и латинскими, машины научились гораздо позже. От тех далеких времен, когда они знали лишь отдельные латинские буквы, они сохранили способность расставлять эти буквы в алфавитном порядке.

Есть в Бейсике функция CHR ⊗( ). С ее помощью выводится на дисплей символ, код которого совпадает с аргументом в скобках. Если выполнить программу

```
FOR I = 65 TO 90: PRINT CHR ⊗(I):NEXT I,
```

на дисплей выйдут в алфавитном порядке прописные латинские буквы от A (ее код 65) до Z (код 90).

Коды позволяют сравнивать по величине и буквы, и слова. Для компьютера слово JOHNSON меньше слова SMITH, так как код буквы J (74) меньше кода буквы S (83). Слово ABAK меньше слова ARAB, поскольку код буквы B (66) не превышает кода буквы R (82), а первые буквы у обоих слов одинаковы. Это позволяет упорядочивать слова так же, как и числа.

Алгоритм упорядочивания числовых массивов известным методом «пузырька» я в свое время уже программировал. Отыскиваю его в своем маленьком фонде алгоритмов и программ. Элементы массива обозначены символом N (I), размер массива — S, в программе еще используются вспомогательные переменные N и P. Вторая из них сначала полагается равной нулю. Потом в цикле (FOR I = 2 TO S) производится попарное сравнение соседних элементов массива на неравенство N (I — — 1) < = N (I). Если неравенство выполняется, индекс I увеличивается на единицу и сравниваются следующие два числа (NEXT I). Если не выполняется, то перед увеличением индекса I элементы N (I — 1) и N (I) меняются местами с помощью вспомогательной переменной N. Делается это так: N = N (I — 1): N (I — 1) = N (I):N (I) = N. После каждой такой перестановки переменной P присваивается значение 1. Когда индекс I возрастает до значения S, происходит проверка: P = 1? Если да, то P вновь становится нулем, и цикл повторяется еще раз, начиная опять-таки с I = 2. Так происходит до тех пор, пока за

время прохождения всего массива переменная P ни на одном шаге не потеряет нулевого значения, а это говорит о том, что ни в одной паре соседних элементов меньший не следует за бо́льшим. На этом упорядочивание массива заканчивается.

Программа, выполняющая такую процедуру, занимает всего три строки:

```
10 P = 0:FOR I = 2 TO S:IF N(I-1)<=N(I) THEN 30
20 N = N(I-1):N(I-1) = N(I):N(I) = N:P = 1
30 NEXT I:IF P = 1 THEN 10
```

Чтобы приспособить ее для упорядочивания литерных массивов, надо лишь приписать к букве N символ литерной переменной — «солнышко». И если бы нашими гостями были Смиты, Джонсоны и Харрисы, то для упорядочивания их фамилий я написал бы такую программу:

```
5  INPUT S:DIM N⊗(S):FOR I = 1 TO S:INPUT N⊗(I):NEXT I
10 P = 0:FOR I = 2 TO S:IF N⊗(I - 1)<=N⊗(I) THEN 30
20 N⊗ = N⊗(I - 1):N⊗(I - 1) = N⊗(I):N⊗(I) = N⊗ :P = 1
30 NEXT I:IF P = 1 THEN 10
40 FOR I = 1 TO S:PRINT N⊗(I):NEXT I
```

По мере повторения цикла, записанного в строке 50, я присваивал бы последовательным элементам массива N ⊗ ( ) значения SMITH, JOHNSON, HARRIS. . . В строке 10 программа сравнивала бы фамилии. Закончив упорядочивание, она переходила бы на строку 40: здесь, тоже в циклическом порядке, она напечатала бы весь список по алфавиту.

С Поповыми и Ивановыми так просто, к сожалению, не получается. Если попросить компьютер, который может выводить на дисплей русские буквы, вывести их с помощью команд

```
FOR I = 96 TO 126:PRINT CHR ⊗(I):NEXT I,
```

то машина выдаст странный на первый взгляд набор: Ю, А, Б, Ц, Д, Е, Ф, Г, Х, И, Й, К, Л, М, Н, О, П, Я, Р, С, Т, У, Ж, В, Ь, Ы, З, Ш, Э, Щ, Ч. В таком порядке увеличиваются коды букв кириллицы от 96 до 126.

Если по этим кодам расставлять имена не английских, а русских мальчиков, то Юра окажется на первом месте, а Валера попадет чуть ли не в самый конец списка.

— Спокойной ночи! — иронически произнесла жена, заглядывая в мою комнату.

— Буквально еще полчасика! — по привычке ответил я умоляющим голосом, хотя прекрасно понимал, что работа мне предстоит еще серьезная и долгая.

Прежде всего машина должна знать алфавитный порядок русских букв. С помощью оператора DATA я запишу в программу русский алфавит, по крайней мере те тридцать букв, с которых могут начинаться фамилии. Массивов мне придется использовать уже не один, а два. Первый — это список фамилий,

обозначим его C ⊗ ( ). Второй — это числа, указывающие, на каком месте в русском алфавите располагается первая буква каждой фамилии, обозначим его N ( ). По элементам второго массива и пойдет упорядочивание. Вести его можно все тем же методом «пузырька».

Характерные для него перестановки нужно будет производить не только в числовом массиве номеров N ( ), но одновременно, притом точно такие же, в литерном массиве фамилий. Значит, наряду со вспомогательной числовой переменной N, которая требуется по методу «пузырька», понадобится еще и вспомогательная литерная переменная C ⊗: она будет хранить значение одного из переставляемых при упорядочивании элементов массива C ⊗ ( ). И еще одна промежуточная литерная переменная понадобится: ей будут присваиваться значения букв, считываемых из русского алфавита. Я обозначил ее B ⊗.

Если придется использовать программу для каких-то серьезных целей (например, для составления списка литературы к научной статье), нужно предусмотреть достаточно максимальную длину переменных C ⊗ ( ) и C ⊗. Пусть она будет равна 200. И позиций в списке пусть будет столько же. А длина переменной B ⊗, естественно, равна единице.

Так вырисовывалось начало программы:

```
10 DIM  C ⊗ (200) 200, C ⊗ 200, N (200),B ⊗ 1
20 DATA «А», «Б», «В», «Г», «Д», «Е», «Ж», «З», «И», «Й»,
        «К», «Л», «М», «Н», «О», «П», «Р», «С», «Т», «У»,
        «Ф», «Х», «Ц», «Ч», «Ш», «Щ», «Ы», «Э», «Ю», «Я»
```

Потом — уже знакомый ввод элементов списка фамилий:

```
30 INPUT «РАЗМЕР СПИСКА», S:FOR  I = 1 TO S
40 PRINT I;« - Я ПОЗИЦИЯ»;: INPUT C ⊗ (I)
```

Дальше — последовательное считывание букв русского алфавита для сравнения их с первой буквой только что введенной фамилии. Номер буквы алфавита, совпавший с первой буквой фамилии, и станет значением элемента N(I).

Я уже начал писать слово READ, но вовремя спохватился. Оператор READ при каждом новом обращении к нему считывает из списка, заданного оператором DATA, все новые элементы, следующие за теми, которые были считаны при предыдущих обращениях. А это совсем не то, что нужно: просмотр русского алфавита каждый раз должен начинаться с первой буквы. И вместо оператора READ я написал RESTORE: он указывает программе вернуться к началу списка, заданного оператором DATA.

Считывание удобно вести в цикле, пусть его счетный индекс обозначается J. Значение считанной буквы присваиваем литерной переменной B ⊗. С помощью функции STR ( ) берем первую букву элемента C ⊗ (I). Обе буквы сравниваем на сов-

падение. Нет совпадения — считываем следующую букву алфавита, есть — присваиваем значение J элементу N (I). Это и значит, что первая буква I-го элемента списка фамилий в русском алфавите стоит под номером J. Как только элемент N (I) получит свое значение, можно переходить к следующей фамилии, наращивая I:

```
50 RESTORE : FOR J = 1 TO 30 : READ B ⊗ :
   IF B ⊗ = STR(C ⊗ (I),1,1)THEN 60 : NEXT J
60 N(I) = J:NEXT I
```

Так все элементы массива C ⊗ (I) получают свои номера N (I):

1. Попов N(1) = 16
2. Иванов N(2) = 9
3. Копылов N(3) = 10
4. Букин N(4) = 2

Я перечитал написанное, подумал и приписал в конце строки 50 два оператора: PRINT «ОШИБКА»: GOTO 40. Если ни одного совпадения B ⊗ = STR(C ⊗ (I),1,1) не обнаружено, значит, первая буква введенной фамилии не из русского алфавита или же сам алфавит задан неверно. Выводя на дисплей слово ОШИБКА, машина попросит повторить ввод.

Дальше — легче. Я переписал алгоритм упорядочивания по методу «пузырька» так, чтобы он располагал по порядку элементы сразу двух массивов — N( ) и C ⊗ ( ). Если у двух последовательных элементов массива C ⊗ ( ) номера убывают (как у элементов ПОПОВ и ИВАНОВ в моем списке), то меняются местами и они, и их номера:

```
70 P = 0:FOR I = 2 TO S:IF N (I − 1) <= N (I) THEN 90
80 C ⊗ = C ⊗ (I − 1):C ⊗ (I − 1) = C ⊗ (I):C ⊗ (I) = C ⊗ :
   N = N (I − 1):  N (I − 1) = N (I):  N (I) = N:P = 1
90 NEXT I:IF P = 1 THEN 70
```

Закончил я программу (рис. 3.5) оператором, который печатает список фамилий в алфавитном порядке:

```
100 FOR I = 1 TO S:PRINT I;«.»; C ⊗ (I):NEXT I
```

— Доброе утро! — сказала жена, заглядывая в мою комнату. Она удивилась моему свежему, радостному виду: до сих пор не верит, что работа с персональным компьютером — это волшебство. Снять усталость после бессонной ночи — далеко не самое удивительное чудо, которое даруется вместе с успешным решением придуманной тобой задачи.

Как увеличить глубину сортировки!

Программа «Алфавит» расставляет литерные переменные только по первой букве. Слова с одинаковыми первыми буквами сохранят тот порядок, в котором они вводились в машину. Если такой частичной сортировки недостаточно, то в программу можно ввести еще один показатель — глубину сортировки, который будет определять, по скольким

```
10 DIM C¤(200)200,C¤200,N(200),B¤1
20 DATA "А","Б","В","Г","Д","Е","Ж","З","И","Й",
        "К","Л","М","Н","О","П","Р","С","Т","У",
        "Ф","Х","Ц","Ч","Ш","Щ","Ы","Э","Ю","Я"
30 INPUT "РАЗМЕР СПИСКА",S:
   FOR I=1 TO S
40 PRINT I;"-Я ПОЗИЦИЯ";:INPUT C¤(I):RESTORE
50 FOR J=1 TO 30:READ B¤:IF B¤=STR(C¤(I),1,1)THEN 60:
   NEXT J:PRINT "ОШИБКА":GOTO 40
60 N(I)=J:
   NEXT I
70 P=0:
   FOR I=2 TO S:
   IF N(I-1)<=N(I) THEN 90
80 C¤=C¤(I-1):C¤(I-1)=C¤(I):C¤(I)=C¤:
   N=N(I-1):  N(I-1)=N(I):  N(I)=N:  P=1
90 NEXT I:
   IF P=1 THEN 70
100 FOR I=1 TO S:PRINT I;".";C¤(I):NEXT I
```

Рис. 3.5. Программа «Алфавит»

первым буквам она ведется. Если этот показатель принять равным размеру самой длинной литерной переменной списка, то ранжировка по алфавиту будет полной.

Составьте такую программу. За основу можно взять вышеописанную, дополнив ее еще одним циклом с параметром, выражающим глубину сортировки. На первое место алфавита при этом нужно поставить символ пробела. Но перед началом такой работы проверьте, каковы коды русских букв вашего компьютера. Если они возрастают в строгом соответствии с алфавитом, то сортировка литерных переменных ничем не будет отличаться от такой процедуры с числовыми переменными.

«Дружественность» Бейсика

Какая сила приковывает человека к персональному компьютеру, заставляя просиживать за ним часами, не отрываясь? Трудно ответить на этот вопрос исчерпывающе. Но можно утверждать без сомнений: столь властное притяжение в значительной мере обусловлено характерными особенностями Бейсика — языка, которым оснащено большинство современных персональных компьютеров. И в самом деле, какой другой язык программирования так близок человеческой речи и символике привычных математических выкладок? Какой еще язык допускает столь широкий спектр режимов работы на компьютере — от реализации весьма сложных программ до непосредственных кратких расчетов, по простоте своей сравнимых с вычислениями на инженерном микрокалькуляторе? В каком другом языке так полно реализован принцип умолчания, когда компьютер при недостатке исходной информации решает сам, как ему поступить?

Бейсик не только прощает человеку многие неточности и промахи, присущие начинающим программистам, но и позволяет свести на «нет» последствия просчетов. Многие ошибки, как мы еще не раз убедимся, часто используются для того, чтобы изменять в нужном направлении последовательность выполнения запрограммированных действий машины. Такими переходами в программе «заведует» так называемый анализатор ошибок. Мы познакомимся с ним в следующей главе.

Этюд 16. В КАКОЙ ДЕНЬ НЕДЕЛИ РОДИЛАСЬ ВАША БАБУШКА?

Я медиевист. Вы, наверное, впервые слышите про такую профессию. Поясню: медиевист — это историк, который занимается средними веками. Но не про средневековье сейчас речь. Я хочу поговорить о фактах новейшей, современной истории. Конкретно — о внедрении вычислительной техники. Все знают, что она сейчас нужна во всех отраслях знания и техники. Почему же в одних она внедряется успешно, а в других — не очень? Например, у авиаконструкторов, экономистов она незаменима, а из лингвистов, врачей ею овладели лишь некоторые. В нашем институте, например, никто, кроме меня, не представляет, какие возможности ЭВМ дает историку: помогает расшифровывать тексты на умерших языках, уточнять толкования документов прошлого, оценивать степень причинно-следственных зависимостей между историческими явлениями и даже моделировать возможное течение отдельных исторических событий при варьировании исходных обстоятельств.

Да и в тех отраслях, где компьютеры вроде бы прижились, дело обстоит по-разному: где-то машины эксплуатируются вовсю, а где-то простаивают. Об этом мы часто читаем в газетах. Лично мне кажется, что здесь все решает человеческий фактор. В любом коллективе должен найтись человек, который первым освоит ЭВМ, покажет всем коллегам, как она помогает в работе, заразит своим энтузиазмом весь коллектив — и, глядишь, дело пошло.

Кстати, у самих застрельщиков компьютеризации увлечение компьютером тоже развивается, как правило, от какой-то ма-

лости. Иногда от ерунды, от пустяка. Сужу опять-таки по себе. Однажды я разговорился со знакомым программистом. Посетовал на то, что мы, историки, совсем не используем математику. Он возразил: «Сказать, что совсем, было бы, пожалуй, неправильно. Откуда, например, вам, историкам, известно, что князь Игорь выступил в последний переход к месту своей роковой битвы с половцами 1 мая 1185 года?» Я говорю: «Год похода известен по летописи, а в день выступления случилось солнечное затмение, оно описано в «Слове о полку Игореве». Он отвечает: «Факт затмения — это еще не дата. Вы обратились к математикам, у них есть формула, которая выражает даты всех солнечных затмений, и они по этой формуле рассчитали, что в 1185 году затмение произошло 1 мая. Так что сказать, что вы совсем не используете математику, нельзя. Другое дело, что вы могли бы использовать ее чаще, лучше. Скажем, есть формулы, по которым можно рассчитывать фазы Луны, время восхода и захода Солнца. Если их запрограммировать, то компьютер за считанные секунды выдавал бы вам такие подробности исторических событий, о которых вы и не подозреваете. Ты знаешь, как выглядел небосвод в Варфоломеевскую ночь? Нет? А я догадываюсь, проводил однажды такой расчет ради развлечения, когда проверял программу для определения фаз Луны. Во время Варфоломеевской ночи было полнолуние. Представляешь ту ужасную резню при ярком, мертвенно-белом лунном свете? . . . Ты и в своей собственной биографии мог бы при помощи компьютера открыть много интересного. В какой, например, день недели родилась твоя бабушка? Не представляешь? А ты сам? Тоже не знаешь? И не можешь, стало быть, вообразить обстановку того «исторического» дня».

С этого разговора все и началось. Мы условились, что он обучит меня Бейсику. А через месяц мы уже сели за составление программы для определения дня недели по дате.

Дни недели чередуются в строгом порядке: за понедельником всегда идет вторник, а за субботой — воскресенье; в каждой неделе ровно семь дней. 31 декабря 1900 года по новому стилю был понедельник — я выяснил это по справочникам. Если по интересующей меня дате я узнаю, сколько дней прошло с того дня, я поделю это число на семь и посмотрю остаток. Если он равен единице, то интересующий меня день — вторник, если двойке — то среда, и так далее. Но узнать число дней, протекших между двумя датами, — дело совсем не простое. Если перед вторником всегда бывает понедельник, то перед первым числом месяца — что угодно: и 28-е, и 29-е, и 30-е, и 31-е. Число дней в году тоже не всегда одно и то же: в високосном — 366, в прочие — 365.

Друг похвалил меня за тщательный анализ задачи:

— Вот ты говоришь: первый день — вторник, второй — сре-

да, третий — четверг... Один, два, три и так далее — это числовой массив. «Вторник», «среда», «четверг» и так далее — литерный. Можно, конечно, и так выводить результаты расчета на дисплей: 1-й день недели, 2-й день недели, чтобы ты сам домысливал их названия. Но лучше придать результату самую понятную форму: «28 января 1941 года — вторник». Дату, которая тебя интересует, ты вводишь сам, а чтобы машина приписала к ней день недели, надо организовать литерный массив из их названий — обозначим его W ⊗ ( ). Если соблюдать твою нумерацию дней недели, то выглядеть этот массив будет так: W ⊗ (1) = «ВТОРНИК», W ⊗ (2) = «СРЕДА», и так далее. Когда программа вычислит номер дня недели D, останется только выполнить команду PRINT W ⊗ (D).

Я остановил его: «Это же явно самая последняя строка программы! Ты начинаешь с конца!»

— И правильно делаю! — возразил он. — Есть хорошая английская пословица: умный начинает с конца, глупый кончает в начале.

— Но умный, — не остался я в долгу, — начав дело с конца, доводит его до начала, не так ли?

— Чтобы ты побыстрее признал меня умником, я сейчас же перепрыгну на начало программы, на первую ее строчку — не возражаешь?

— На ту строчку, где вводится дата?

— Нет, еще раньше надо организовать массив W ⊗ ( ), потому что он будет один на все даты. Да ты и сам сможешь это сделать!

— Циклом?

— Циклом! Точно!

Я написал довольно уверенно:

```
10 DATA «ВТОРНИК», «СРЕДА», «ЧЕТВЕРГ», «ПЯТНИЦА»,
   «СУББОТА», «ВОСКРЕСЕНЬЕ», «ПОНЕДЕЛЬНИК»
20 FOR D = 1 TO 7:READ W⊗(D):NEXT D
```

И продолжил:

```
30 INPUT «ВВЕДИТЕ ДАТУ», A ⊗
```

Он прервал меня:

— Стоп! В каком виде ты собираешься ее вводить?

— В самом естественном: 28 января 1941 года.

— Самом естественном для тебя? Не уважаешь ты своего электронного помощника! Он-то позаботился о том, чтобы быть понятным для тебя, он выводит информацию на дисплей так, чтобы ты не ломал над ней голову. С твоей стороны было бы хорошо, если бы ты вводил в него информацию так, чтобы он перерабатывал ее без лишних затруднений. Можно ведь ту же самую дату записать так: 28.01.41.

— Согласен.

Друг пристально посмотрел на меня: «Не нравится мне, что ты так легко соглашаешься с моим предложением. Ты, наверное, не предвидишь, что из такой записи совсем не просто выделить число дней, месяцев и лет».

— А чего тут сложного? Первые две цифры — число дней, вторые — месяцев, третьи — лет. Сначала с помощью функции STR надо «вырезать» первые две цифры. . .

— Функции STR? Но ведь ее значения — переменные литерные! Когда ты вырежешь первые две цифры из даты, это будет литерная переменная "28", а не число 28. Кстати, когда впоследствии ты станешь вырезать число месяцев, учти, что обозначающие его цифры стоят на четвертой и пятой позициях. Точка — это такая же полноправная деталь даты, как и цифры, она занимает отдельную позицию.

Я повинился в своей невнимательности и дал ему возможность блеснуть эрудицией. «Литерная переменная, — произнес он менторским тоном, — преобразуется в число с помощью функции CONVERT (превращать)». И написал:

```
40 CONVERT STR(A⊗,1,2) TO D:CONVERT STR (A⊗,4,2)
   TO M:CONVERT STR(A⊗,7,2)TO G
```

Я недоверчиво прочел написанное: «Ты ничего не напутал? У тебя число дней в дате обозначено той же буквой, что индекс массива W ⊗ (D)».

— Не напутал, — успокоил он меня, — а сделал специально. Зачем усложнять программу лишними переменными? Я прибавляю к числу D (число дней в заданной дате) число дней в месяцах, прошедших до нее, и результат опять обозначу D. Потом добавлю к нему число дней в годах, прошедших до указанного в дате, и опять обозначу результат буквой D. Это будет число дней, прошедших с 31 декабря 1900 года. Потом я поделю это D на 7, от частного возьму целую часть, вычту ее из D и снова обозначу результат буквой D. Это будет та самая величина, по которой я найду значение W ⊗(D). Понял?

Я понял все лишь тогда, когда он растолковал мне еще раз.

— Учти такой прием экономии на будущее, — посоветовал он. — Давай-ка займемся теперь подсчетом дней по номеру месяца, указанному в дате. Речь идет, как ты понимаешь, о числе дней, прошедших с начала года до начала указанного месяца. Если это январь, то оно равно нулю, если февраль, то оно равно 31, если март, то 31 + 28, т. е. 59, если апрель, то 31 + 28 + 31, т. е. 90. . .

— А если год високосный? Тогда в феврале не 28 дней.

— Это мы исправим прибавлением единицы после проверки, делится ли число лет на 4. А пока сделаем так, как я предлагаю: по числу месяцев в дате организуем в программе прибавление к числу D либо нуля, либо 31, либо 59, либо 90. . ., ну и так далее.

— Прибавление нуля? Зачем это? Лучше написать так:

```
50 IF M = 1 THEN 60
```

Ничего не прибавляя, перейдем на следующую строку.

— Ты прав, благодарю за поправку. Но одиннадцать прочих вариантов остаются.

— Написать одиннадцать подпрограмм? А переходить на них как?

— Так же, как мы передавали бы управление на различные участки программы в зависимости от числа M:

```
50 IF M = 1 THEN 60:M = M — 1:
   ON M GOSUB 5001,5002,5003,5004,5005,5006,5007,
   5008,5009,5010,5011:

5001 D = D + 31:RETURN
5002 D = D + 59:RETURN
5003 D = D + 90:RETURN
5004 D = D + 120:RETURN
5005 D = D + 151:RETURN
5006 D = D + 181:RETURN
5007 D = D + 212:RETURN
5008 D = D + 243:RETURN
5009 D = D + 273:RETURN
5010 D = D + 304:RETURN
5011 D = D + 334:RETURN
```

— Ты не устал? — улыбнулся я. — Давай-ка я сам закончу программу. Число дней в (G — 1) годе, прошедшем до года G, — это 365 * (G — 1) плюс столько дней, сколько было до этого високосных годов, т. е. INT((G — 1)/4). Да еще плюс единица, если год в дате високосный и, значит, G/4 — целое. Это чтобы учесть, что в феврале этого года 29 дней:

```
60 D = D + 365 * (G — 1) + INT ((G — 1)/4):
   IF G/4 > INT (G/4) THEN 70:D = D + 1
```

Он молча следил, как я выписываю эту строку. Когда я кончил, сказал:

— Перечти-ка внимательно, что ты написал. Не замечаешь ошибку? Нет? Прибавлять единицу надо только в том случае, если интересующая тебя дата перевалила за февраль. Не забудь, что из числа месяцев M мы вычитали единицу.

— Действительно, виноват, — еще раз покаялся я и дополнил без того длинную строку:

```
60 . . .: IF G/4 > INT (G/4) OR M < 2 THEN 70: D = D + 1
```

В последних строках я уже не сомневался: вычисляем остаток от деления D на семь:

```
70 D = D — 7 * INT(D/7)
```

Узнаем по этому числу название дня недели и выводим его на дисплей вместе с датой:

```
80 PRINT A⊗;«—»; W⊗(D)
```

— Все?

Он, конечно, не мог допустить, чтобы последнее слово оставалось не за ним: «Нет, не все! А если ты захочешь обратиться к компьютеру с новой датой?» И он дополнил последнюю строку:

```
80 . . .: GOTO 20
```

— Ну, теперь-то все? — снова спросил я.

Друг поглядел на меня с упреком:

— Нет, я вижу, ты еще недостаточно постиг работу программиста. Каждую программу можно усовершенствовать. Посмотри на строку 60 — ее можно записать по-другому, короче, а результат будет тот же:

```
60 D = D + 365 * (G - 1) + INT(G/4):IF M > 1 THEN 70:D = D - 1
```

Можно расширить программу, чтобы она не только выясняла день недели по дате, но еще определяла, какой день недели будет через заданное число дней от сегодняшнего, сколько дней прошло между двумя заданными датами. Тогда сразу после описания массива W ⊗ ( ) запишем с помощью оператора выбора ON. . .GOTO так называемое меню — перечень услуг, какие машина может нам оказать:

```
20 INPUT «1 — ДЕНЬ НЕДЕЛИ ЛЮБОЙ ДАТЫ, 2 — ДАТА И ДЕНЬ
        НЕДЕЛИ ЛЮБОГО ЧИСЛА ПЛЮС (МИНУС) ДНИ,
        3 — ЧИСЛО ДНЕЙ МЕЖДУ ДВУМЯ ДАТАМИ, 4 — КОНЕЦ», A:
   ON A GOTO. . .
```

В зависимости от того, какое число ты введешь, таким станет число A. В зависимости от числа A программа перейдет на тот адрес, который стоит после слова GOTO на месте под номером A. С соответствующего адреса ты запишешь фрагмент программы, который выполняет нужный расчет. Можно еще устроить в программе «защиту от дурака», чтобы не проходили ляпы вроде 33.13.88: тогда пусть машина выведет на дисплей слово ОШИБКА и тем самым попросит тебя повторить ввод. Можно отшлифовать программу до предельной степени логичности, чтобы, например, в списке DATA дни недели перечислялись не со вторника, а, например, с воскресенья. Можно поискать лучшие алгоритмы для определения номера дня на хронологической оси двадцатого века. Если хочешь поупражняться в программировании — попробуй составить такую программу!

Подпрограммы и процедуры

Если вам не удалось составить свой вариант расширенной программы «Вечный календарь», каким он описан в конце диалога историка и математика, разберите вариант, помещенный на рис.3.6. Обратите внимание на номера строк, содержащих подпрограммы обработки календарных дат. То, что эти номера так велики, не случайно. Имена переменных (D 5, например) тоже подобраны в них специально. Дело в том, что одна из этих подпрограмм будет использоваться далее в программе «Карто-

```
10 DEF FNW(D)=1.5+D-7*INT(D/7):FOR D=1 TO 7:READ
   W$(D):NEXT D:DATA "ВОСКРЕСЕНЬЕ","ПОНЕДЕЛЬНИК",
   "ВТОРНИК","СРЕДА","ЧЕТВЕРГ","ПЯТНИЦА","СУББОТА"
20 INPUT "1-ДЕНЬ НЕДЕЛИ ЛЮБОЙ ДАТЫ,2-ДАТА И ДЕНЬ НЕДЕЛИ
   ЛЮБОГО ЧИСЛА ПЛЮС (МИНУС) ДНИ,3-ЧИСЛО ДНЕЙ МЕЖДУ
   ДВУМЯ ДАТАМИ,4-КОНЕЦ",A:ON A GOTO 30,30,50:STOP
30 PRINT"ДАТА";:GOSUB 5001:PRINT W$(FNW(D)):
   IF A=1 THEN 20
40 D0=D:INPUT "ЧИСЛО ДНЕЙ",D:D=D+D0:GOSUB 5007:PRINT
   D$;W$(FNW(D)):GOTO 20
50 PRINT "ДАТА 1";:GOSUB 5001:D0=D:PRINT "ДАТА 2";:
   GOSUB 5001:PRINT D-D0;" ДНЕЙ.":GOTO 20
5000 PRINT "ОШИБКА"
5001 INPUT" ГГ.ММ.ДД ",D$:CONVERT STR(D$,1,2) TO G:
     CONVERT STR(D$,4,2) TO M:IF M>12 THEN 5000:
     CONVERT STR(D$,7,2) TO D:IF D>31 THEN 5000
5002 ON M GOTO 5005,5003,5005,5004,5005,5004,5005,5005,
     5004,5005,5004,5005
5003 IF D>29 THEN 5000:IF G/4>INT(G/4) AND D>28THEN5000
5004 IF D>30 THEN 5000
5005 M=M+1:IF M>3 THEN 5006:M=M+12:G=G-1
5006 D=INT(365.25*G)+INT(30.6001*M)+D:RETURN
5007 G=INT((D-122.1)/365.25):M=INT((D-INT(365.25*G))/
     30.6001):D5=D-INT(365.25*G)-INT(30.6001*M):M=M-1:
     IF M<13 THEN 5008:M=M-12
5008 IF M>2 THEN 5009:G=G+1
5009 D$="00.00.00":CONVERT G TO STR(D$,1,2),(##):
     CONVERT M TO STR(D$,4,2),(##):
     CONVERT D5 TO STR(D$,7,2),(##):RETURN
```

Рис. 3.6. Программа «Вечный календарь»

тека приказов». Поэтому нужно уже сейчас позаботиться о том, чтобы у программ и прилагаемых к ним подпрограмм не совпали ни номера строк, ни имена задействованных переменных. В Бейсике это может привести к существенным сбоям.

Из-за этого, в частности, многие программисты не считают Бейсик «серьезным» языком программирования. Такие алгоритмические языки, как Фортран, Алгол, Паскаль, ПЛ/1, Ада, оперируют не подпрограммами, а *процедурами*, которые более независимы от программ. С помощью процедур можно довольно легко составлять большие программы из готовых блоков-модулей. Это называется *программированием сверху вниз*.

Не вдаваясь в строгое определение процедуры, скажем лишь, что она отличается от подпрограммы двумя особенностями.

Во-первых, процедура вызывается не по номеру первой строки, а по ее имени.

Во-вторых, процедура может иметь свои собственные, так называемые *локальные переменные*, которые, если и совпадут по имени с переменными основной программы и других процедур, с ними не смешаются. Поэтому программисту совсем не обязательно вникать в работу процедуры — ему достаточно только знать ее имя и то небольшое количество переменных, одни из которых примут значения исходных данных перед вызовом процедуры, а другие будут выражать результат ее выполнения. Такие переменные называют *глобальными*.

В нашей подпрограмме определения календарной даты по номеру (для строки 5007 — 5009) глобальными переменными можно назвать D (аргумент), и D ⊗ (функция). Переменные G, M и D5 — локальные. В основной программе их использовать нельзя, так как после каждого вызова подпрограммы значения этих переменных будут изменяться. Подобное ограничение существенно затрудняет использование такой эффективной методики, как структурное программирование сверху вниз. Программу на Бейсике, как правило, всю с начала до конца пишет один человек, начиная с разработки отдельных подпрограмм. Это называется *программированием снизу вверх*.

Какие программы еще можно составить

Есть программы, разработка которых не минует ни одного начинающего программиста. Их список несложно составить, просмотрев учебники по программированию:

решение квадратного уравнения;
решение кубического уравнения;
нахождение факториала числа;
поиск чисел Фибоначчи;
поиск простых чисел;
определение среднего арифметического и среднего геометрического ряда чисел;
сортировка массива чисел быстрыми методами;
нахождение корней алгебраического уравнения методами Ньютона, половинного деления и секущих;
решение системы линейных алгебраических уравнений;
поиск максимума одномерной функции;
построение гистограмм;
решение обыкновенных дифференциальных уравнений методами Эйлера и Рунге — Кутта;
построение магического квадрата;
составление «вечного календаря»;
определение биологических ритмов состояния человека;
игра «крестики-нолики»;
игра «морской бой»;
расстановка восьми ферзей на шахматной доске.

Три из этих программ мы уже составили. Попробуйте, если хотите, составить какие-нибудь из остальных.

Этюд 17. МОЮ СЕКРЕТАРШУ ЗОВУТ «ИСКРА»

По специальности я радиоинженер, а по должности — заведующий лабораторией. Мы занимаемся оптимизацией радиотехнических устройств некоторого класса. Расчеты ведем на «Искре 226». Но гораздо бóльшую часть времени мне приходится отводить административной деятельности. Из-за этого мой рабочий день нередко бывает заполнен не диалогом с умной машиной, а различными совещаниями, встречами, телефонными звонками. Секретарь мне нужен «позарез», но, видите ли, не положен по штату.

Я решил поручить функции секретаря своей «Искре». Много от нее не требуется. В понедельник и четверг (на эти дни мы в лаборатории стараемся «свалить» все организационные дела) машина утром должна опросить меня и сотрудников о планах на день, затем в течение дня «держать» на дисплее точное время и информацию о ближайшем или текущем деле и подавать звуковые сигналы в моменты их начала и окончания: «Внимание, мол, Николай Петрович! Не забыли ли вы позвонить кому следует? Не пропустите ли совещание? Успеете ли отправить в главк отчет об эффективности использования ЭВМ во вверенном вам подразделении?» И все такое прочее.

Предложил я своим сотрудникам составить такую программу. Составили. Я ее внимательно изучил. Хорошая программа, короткая — всего 18 строк (рис. 3.7). Из них две последние — подпрограммы для счета времени. Остальные группируются

```
10 DIM A¤(50)200,A¤200,S(2,50):I=0:PRINT"ТОЧНОЕ ВРЕМЯ";
   :GOSUB 170:S0=S:GOSUB 180:T0=T
20 IF I=50 THEN 100
30 I=I+1:PRINT I;"-ОЕ ЗАДАНИЕ (CR/LF-ВСЕ)";:INPUT
   A¤(I):IF A¤(I)=" " THEN 100
40 PRINT"НА КАКОЕ ВРЕМЯ НАЗНАЧЕНО";:GOSUB 170:S(1,I)=S:
   GOSUB 180:IF S(1,I)>S THEN 50:PRINT"ПОЗДНО.":GOTO 40
50 INPUT "ПРОДОЛЖИТЕЛЬНОСТЬ(МИН)",S:S(2,I)=S(1,I)+60*S:
   IF I=1 THEN 20
60 FOR J=1 TO I-1:REM ПРОВЕРКА.НЕТ ЛИ НАКЛАДКИ
70 IF S(1,I)>=S(2,J) OR S(2,I)<=S(1,J) THEN 90
80 PRINT "НА ЭТО ВРЕМЯ УЖЕ НАЗНАЧЕНО - ";A¤(J):GOTO 40
90 NEXT J:GOTO 20:REM ЗАПРОС ИНФОРМАЦИИ ПО НОВОМУ ДЕЛУ
100 P=0:FOR J=2 TO I:IF S(1,J-1)<=S(1,J) THEN 110:
    FOR K=1 TO 2:S=S(K,J-1):S(K,J-1)=S(K,J):S(K,J)=S:
    NEXT K:A¤=A¤(J-1):A¤(J-1)=A¤(J):A¤(J)=A¤:P=1
110 NEXT J:IF P=1 THEN 100:REM СОРТИРОВКА НЕ КОНЧИЛАСЬ
120 FOR J=1 TO I:PRINT HEX(03):PRINT:FOR K=1 TO 2:PRINT
    INT(S(K,J)/3600);" ЧАС.";S(K,J)/60-60*INT(S(K,J)/
    3600);" МИН.";:IF K=2 THEN 130:PRINT "- ";
    STR(A¤(J),1,LEN(A¤(J)));" -";
130 NEXT K
140 GOSUB 180:PRINT HEX(01);"ВРЕМЯ ";:PRINTUSING "##.",
    H;M;S-3600*H-60*M:IF ABS(S(1,J)-S)>=5 AND
    ABS(S(2,J)-S)>=5 THEN 150:PRINT HEX(07)
150 IF S<S(2,J) THEN 140:REM J-Е ДЕЛО НЕ ЗАКОНЧИЛОСЬ
160 NEXT J:STOP "НА СЕГОДНЯ ВСЕ.ДО ЗАВТРА."
170 INPUT "(ЧАСЫ,МИНУТЫ)",H,M:IF H>23 OR M>59 THEN 170:
    S=3600*H+60*M:RETURN:REM ЗАПРОС ВРЕМЕНИ У ЧЕЛОВЕКА
180 INPUT ¤T:S=S0+(T0-T)/2000:H=INT(S/3600):
    M=INT(S/60-60*H):RETURN:REM ЗАПРОС ВРЕМЕНИ У МАШИНЫ
```

Рис. 3.7. Программа «Секретарша»

в короткие блоки, те делятся на еще более короткие кусочки — по две-три строки в каждом. Смысл этих блоков и кусков прост. Тем более, что он поясняется толковыми ремарками. Да и фразы, которые вставлены в программу для вывода на дисплей, тоже способствуют пониманию. А они стоят чуть ли не в каждой третьей строке.

Когда программа делится на такие четкие части, когда она снабжена ясными примечаниями, то человеку, хорошо знающему Бейсик, читать ее легко и приятно. Понять ее можно даже с минимальными знаниями языка. Говорю так с полной уверенностью, потому что уже не раз демонстрировал и разъяснял эту программу заведующим других отделов, которые прослышали про моего электронного секретаря.

Строка 10. Здесь задаются массивы и важнейшие переменные, используемые в программе. Массив A ⊗ (50)200 — список мероприятий дня. Их должно быть не больше пяти-

десяти. Компьютер, конечно, может хранить информацию и о большем количестве дел, но кто это все выполнит?! На запись каждого отводится 200 знаков — этого вполне достаточно. A ⊗ — вспомогательная литерная переменная, она используется при расстановке запланированных мероприятий в хронологическом порядке.

Теперь числовой массив S (2,50). Лучше объяснить его, разбив на две части. S (1,50) — данные о времени начала каждого из запланированных дел, S (2,50) — о времени окончания. В памяти машины эти сроки записаны в секундах, прошедших от полуночи, на дисплей выводятся в привычном виде — часы, минуты, секунды.

В машину время вводится тоже в привычном виде: сначала часы, потом через запятую — минуты. Эти значения присваиваются переменным H и M. Из них формируется число S — значение времени в секундах, которое и записывается в память машины.

В строке 10 встречается еще величина i. Это счетный индекс. В первой же строке ему придается нулевое значение. На дисплей выводится надпись ТОЧНОЕ ВРЕМЯ. Как только она появилась, управление передается на подпрограмму, записанную в строке 170. Здесь машина останавливается и выводит на дисплей вопрос: ЧАСЫ, МИНУТЫ? В ответ вы должны набрать на клавиатуре через запятую значения ближайших часов и минут (8,45 или нечто подобное), а в момент, когда секундная стрелка дойдет до цифры 12, нажать клавишу ВК. Переменная S примет значение числа секунд, прошедших от полуночи до момента выставления на машине точного времени.

Когда программа вернется на строку 10, это же значение будет придано переменной S0, затем — переход на другую подпрограмму, записанную в строке 180. Тут в самом начале есть хитрая команда INPUT ⊗ T. Она автоматически запрашивает у встроенных в машину часов значение переменной T, которая при запуске программы имеет некое астрономическое значение, а с каждой двухтысячной долей секунды убывает на единицу. Следующие операторы строки 180 переводят машинное время в привычные для нас часы, минуты и секунды.

Кстати, в строке 170 устроена защита от неверных действий пользователя: если введенное число часов превышает 23, а число минут — 59, машина попросит повторить ввод.

Строка 20 — тоже защитная: если число планируемых мероприятий достигло 50, машина прекращает их прием и переходит к упорядочиванию их списка, на строку 100. Но о ней позже, в свое время.

Строки 30 — 90. Диалог машины с человеком, который сообщает ей свои планы на день.

В строках 30 — 50 компьютер запрашивает формулировку очередного задания (строка 30), время, на которое оно назначено (строка 40), продолжительность (строка 50). Все это вы должны набрать на клавиатуре.

В строках 60 — 90 машина предотвращает возможные накладки. На дисплее еще горит последняя введенная вами строчка, а машина не торопится запрашивать новое задание. Компьютер обдумывает про себя, не «наползает» ли только что запланированное вами I-е дело на какое-то J-е из уже введенных. Проверка ведется в цикле: FOR J = 1 TO I—1. Машина следит, во-первых, чтобы начало I-го дела не оказалось раньше конца J-го, S (1,I) >= S (2,J); во-вторых, чтобы конец I-го дела не оказался позже начала J-го, S(2,I)<= S(1,J). Оба условия — в строке 70.

Если подобное все-таки случится, не беда. «Искра» вежливо сообщит: НА ЭТО ВРЕМЯ УЖЕ НАЗНАЧЕНО (строка 80), назовет мероприятие A ⊗ (J), назначенное на это время, перейдет на строку 40 и тут вновь запросит время начала дела, о котором ведется разговор.

Ревностно следит компьютер и за тем, чтобы ничего не планировалось на уже прошедшее время (конец строки 40).

Обратите внимание: в строке 30, в тексте, который выводится на дисплей, в скобках есть примечание CR/LF — ВСЕ. Это напоминание о порядке прекращения диалога. Если вы желаете его закончить, то при очередном вопросе машины ничего на клавиатуре не набирайте, а просто нажмите клавишу CR/LF*. Компьютер тогда сработает по условному оператору IF A ⊗ (I) = « »THEN 100, т. е. перейдет на строку 100.

Строки 100 — 110. Упорядочивание списка дел. Информацию о делах предстоящего дня лучше вводить в ЭВМ в порядке их важности, не обращая внимания на хронологию. Так мы обычно и поступаем, заполняя чистую страницу ежедневника. Хронологический порядок в этом списке машина наведет сама по известному методу «пузырька». ЭВМ сравнивает каждые два соседних события, и если первое должно произойти позже второго, то они меняются местами. Распорядок дня «утрясается» до тех пор, пока все мероприятия не выстроятся в должном порядке. Тогда машина передает управление на строку 120.

Строки 120 — 160. Монолог ЭВМ, которая поглядывает на часы и сообщает о ближайшем или текущем мероприятии дня.

---

* В компьютере, для которого составлена разбираемая программа, соответствует клавише ВК.

Здесь работают операторы, для которых в Бейсике нет служебных слов — есть только коды. В программе они записываются с помощью оператора вида PRINT HEX( ). В скобках — код нужного действия: 01 — перевод курсора в левый верхний угол дисплея, 03 — то же, с очисткой дисплея, 07 — подача звукового сигнала. Хорошо, если бы «Искра» могла разговаривать и сообщать об очередном деле прямым текстом (кстати, такие машины уже существуют). Но она на это не способна и только подает зуммер в момент начала и окончания дел.

По началу строки 120 видно, что этот фрагмент программы представляет собой цикл. FOR J = 1 TO I: дела перебираются от первого до последнего по времени. PRINT HEX (03): дисплей очищен, курсор в левом верхнем углу. PRINT (без всяких последующих символов): курсор сдвинулся строкой ниже. FOR K = 1 TO 2: при K = 1 печатается время начала очередного дела S (1, J), его формулировка A ⊗ (J), при K = 2 — время его окончания S (2,J).

Строка 140 начинается с обращения к подпрограмме, записанной в строке 180. Время S, прошедшее с полуночи, вычисляется в секундах, переводится в часы и минуты — H и M. PRINT HEX (01); «ВРЕМЯ»: курсор опять в левом верхнем углу; здесь появляется слово ВРЕМЯ; текст, который до этого был выведен на дисплей, сохраняется. Оператор PRINT USING задает формат вывода данных о времени: две цифры и за ними точка. Так вслед за словом ВРЕМЯ появляются три пары цифр: часы, минуты и секунды. Далее проверка двух неравенств, соединенных логическим союзом AND, по-русски И. В них текущее время S сравнивается со временем начала и конца очередного мероприятия. Если абсолютная величина их разности превышает 5 секунд, то управление передается на следующую, строку 150. Если хотя бы одна из разностей по абсолютной величине меньше 5 секунд, строка выполняется до конца: PRINT HEX (07). Звучит зуммер. Пока время не подошло к сроку окончания очередного дела (строка 150), управление вновь и вновь передается на строку 140, где опять и опять запрашивается текущее время. Подошло — индекс J увеличивается на единицу (строка 160), происходит переход на строку 120, где на дисплей выводится информация о следующем деле.

Когда индекс J превышает I — число запланированных дел, происходит выход из цикла, машина останавливается (STOP), и на дисплее загорается: НА СЕГОДНЯ ВСЕ. ДО ЗАВТРА.

С обязанностями секретаря моя «Искра» справляется неплохо. Ребята шутят: «Скоро Н П научит свою секретаршу чай заваривать и бутерброды подавать».

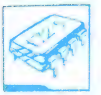
Как усовершенствовать программу

Программа «Секретарша» (см. рис. 3.8) разбита на два участка — участок диалога машины с человеком, который сообщает ей свои планы на день, и участок монолога ЭВМ, которая следит за временем и выводит на дисплей нужную информацию.

Предлагаем читателям так изменить программу, чтобы человек имел возможность в любой момент прервать монолог машины и вернуться к начальному диалогу.

Новая программа должна обеспечить возможность вносить изменения в список запланированных дел. Это важно как на тот случай, если какое-то из них было забыто при составлении списка, так и для того, чтобы можно было переносить на более поздний срок мероприятие, выполнить которое в настоящий момент невозможно. Для подобного усовершенствования программы в цикле монолога нужно предусмотреть команды опроса клавиатуры, которые при нажатии человеком, например, клавиши «С» (стоп) передали бы управление программой в начало участка диалога. Подобный прием применялся в программе «Таблица умножения» (см. рис. 2.3).

Режим реального времени



Программа «Секретарша» (см. рис. 3.8) — единственная в книге, выполняемая в режиме реального времени, когда машина совершает нужные действия, ориентируясь на показания встроенных часов. В таком режиме работают все ЭВМ, управляющие технологическими процессами или собирающие и обрабатывающие данные, поступающие с экспериментальных стендов.

От всех остальных программ книги требовалось качество хорошего бегуна — с поставленными задачами ЭВМ должна была управиться за минимальное время. Программа, работающая в режиме реального времени, подобна участнику автомобильного ралли, которому нужно прибыть на контрольный пункт в нужное время — не раньше и не позже.

«Искра 226» ориентируется во времени с помощью оператора INPUT ⊗. В других версиях Бейсика текущее время выражают системные переменные TI, TIME и т. д. Есть переменные, хранящие и календарные даты. Наиболее совершенные персональные компьютеры снабжены батарейкой, которая после отключения электропитания машины не дает остановиться встроенным часам с календарем. Такие часы после включения машины не нужно выставлять, как это делалось при запуске программы «Секретарша» на «Искре 226».

Если ваш компьютер не имеет встроенных часов, то для их имитации можно организовывать циклы, время однократного выполнения которых известно. Так было сделано при составлении программы «Таблица умножения» (см. рис. 2.3).

ЭТЮД 18. «ХРАНИ МЕНЯ, МОЙ ТАЛИСМАН . . .»

На моем дисплее — программа для составления частотного словаря литературных текстов. Ее написал знакомый программист. Мы вместе листали Словарь языка Пушкина. В четырех увесистых томах были перечислены все слова, которые поэт использовал в своих произведениях, и указано, сколько раз.

— Ты представляешь себе, как составлялся этот четырехтомник? — спросил меня друг.

Я вообразил эту работу, читая по памяти стихотворение А. С. Пушкина «Храни меня, мой талисман . . .». В первых двух строках этого произведения — восемь слов, не считая предлога «во». Берем восемь карточек и вписываем в них эти слова.

Далее строка за строкой (в этом стихотворении их 20), стихотворение за стихотворением. И каждое новое слово — это новая карточка. Слово, уже встречавшееся ранее, — это новая галочка на имеющейся карточке.

Как же нужно любить великого поэта, чтобы решиться на такой труд! В программе, способной это выполнить, всего 17 строк (рис. 3. 8). Вчитываюсь в них. В строке 20 — перечень того множества символов, чьим взаимодействием образуется будущий словарь. S ⊗( )— строки анализируемого текста, W ⊗ ( )— слова, занесенные в частотный словарь, W%( )— частота их встречаемости в тексте,

```
10 INPUT «ЧИСЛО СТРОК В ТЕКСТЕ?»; N
```

Ввожу число строк стихотворения — 20.

```
30 FOR I = 1 TO N : LINPUT S ⊗ (I) : NEXT I
```

```
10 INPUT "ЧИСЛО СТРОК В ТЕКСТЕ?";N
20 DIM S$(N),W$(10*N),W%(10*N)
30 FOR I=1 TO N:LINPUT S$(I):NEXT I
40 J=0:L=0
50 FOR I=1 TO N:REM ПЕРЕБОР СТРОК ТЕКСТА
60 S$=S$(I):S=0:E=0:REM S-НАЧАЛО,E-КОНЕЦ СЛОВА
70 E=E+1:IF MID$(S$,E,1)>="A" GOTO 70
80 IF E=S+1 THEN 130:REM ДВОЙНОЙ ЗНАК ПРЕПИНАНИЯ
90 W$=MID$(S$,S+1,E-S-1):J=J+1:K=0
100 K=K+1:IF W$<>W$(K) AND K<=L GOTO 100
110 IF K<=L THEN
      REM НОВОЕ СЛОВО:      W%(K)=W%(K)+1:GOTO 130
120 L=K:W$(L)=W$:W%(L)=1  :REM СТАРОЕ СЛОВО
130 S=E:REM ПЕРЕХОД К СЛЕДУЮЩЕМУ СЛОВУ
140 IF E<LEN(S$) GOTO 70:REM СТРОКА НЕ КОНЧИЛАСЬ
150 NEXT I:REM ПЕРЕХОД К НОВОЙ СТРОКЕ
160 PRINT "ВСЕГО";J;"СЛОВ,ИЗ НИХ В ТЕКСТЕ"
170 FOR K=1 TO L:PRINT W$(K);W%(K):NEXT I
```

Рис. 3.8. Программа «Частотный словарь»

Ввожу первую стихотворную строку, набираю ее буква за буквой на клавиатуре:

ХРАНИ МЕНЯ, МОЙ ТАЛИСМАН,

Нажимаю клавишу ВК и ввожу вторую строку:

ХРАНИ МЕНЯ ВО ДНИ ГОНЕНЬЯ,

И так — до конца стихотворения, до 20-й строки.

40 J = 0 : L = 0

Что значат эти J и L? Спасительные в подобных случаях ремарки начинаются в программе лишь на строке 50. Быть может, нечто разъясняющее встретится далее: «Всего»; J; «СЛОВ. ИЗ НИХ В ТЕКСТЕ» Ясно: J — это счетчик просмотренных слов текста. Ну, а L? Запасусь терпением до конца: смысл отдельного знака не понять, не уяснив всей программы в целом.

50 FOR I = 1 TO N.

И где-то далеко внизу, на строке 150 — NEXT I.

Программа начинает перебор стихотворных строк. Я начинаю перебор строк программы:

60 S ⊗ = S ⊗ (I) : S = 0 : E = 0
70 E = E + 1 : IF MID ⊗ (S ⊗, E, 1) > = «A» GOTO 70

Ремарки подсказывают: S — начальная граница слова, E — граница, отмечающая его конец. Как легко увидеть эти границы нам! И как непросто научить такому видению машину!

Начало и конец слова отмечены либо пробелом, либо знаком препинания. Как будет искать их машина? E = E + 1. На единицу

увеличился номер просматриваемой позиции в стихотворной строке: MID ⊗ (S ⊗, E, 1). Извлечен знак, стоящий на этой позиции. IF MID ⊗ (S ⊗, E, 1) >= «А». Сравнивать по величине знаки, символы, буквы? Это неравенство запутало бы меня вконец, если бы мой друг, вручая мне программу, не обмолвился, что у каждого из символов, имеющихся на клавиатуре компьютера, есть числовой код, так что их можно сравнивать согласно кодам числа. Коды знаков-разделителей меньше кода латинской буквы А. И если неравенство выполняется, значит, извлеченный символ — не пробел и не знак препинания. Просмотр слова будет продолжен, переходим на строку 70.

Быть может, и следующие строки стоит просматривать вместе с той сетью логических переходов, которая тянется от них?

```
80 IF E = S + 1 THEN 130
130 S = E
140 IF E < LEN (S ⊗ ) GOTO 70
```

Читаю условие E = S + 1. Догадался бы я без подсказки, что это намек на соседство двух знаков-разделителей? Возможно. Но сейчас поверю комментарию:

```
90 W ⊗ = MID ⊗ (S ⊗, S + 1, E — S — 1) : J = J + 1 : K = 0
100 K = K + 1
```

Судя по символу W ⊗, здесь из строки стихотворения вырезается очередное слово, начиная с его первой буквы.

Довериться догадке? Нет, буду разбирать до конца: программу не понять целиком, не уяснив смысл каждого символа.

В просмотренном тексте первая буква каждого нового слова стоит на следующей позиции после начального разделительного знака S, т. е. на позиции S + 1. Именно такое число и записано в скобках вторым. А длина слова должна выражаться через E, номер позиции, заключительного разделительного знака. Каким же должно быть такое выражение? E — S? Проверяю этот вариант на примерах и уточняю его: E — S — 1. Все верно — так и записано в скобках. J = J + 1. Счетчик найденных слов увеличился на единицу. K = 0 и в следующей строке K = K + 1. Начинается перебор слов, имеющихся в словаре. Когда же он кончится или прервется? Тогда, когда не выполнится одно из условий, записанных на строке 100:

```
IF W ⊗ <>W ⊗ (K) AND K<= L GOTO 100
```

т. е. когда извлеченное из стихотворной строки слово W ⊗ совпадет с очередным словом из словаря W ⊗ (K) или . . .

И словно озарение: я понял, наконец, смысл переменной L! Это текущее число слов, имеющихся в словаре во время его составления. И строки программы предстают передо мною кристаллически стройной логической структурой. Одно за другим просматриваются слова в словаре:

100 K = K + 1

и если очередное слово W ⊗ (K) не совпадает с анализируемым W ⊗:

IF W ⊗ <>W ⊗ (K)

а слова, имеющиеся в словаре, не просмотрены до конца:

AND K< = L

только тогда для просмотра берется новое слово из словаря:

GOTO 100

Но если не выполнится хотя бы одно из условий строки 100, компьютер переходит к выполнению следующих строк. Здесь надо прежде всего разобраться: по какой из двух причин совершился этот переход? Если потому, что слово из словаря совпало с анализируемым словом:

W ⊗ = W ⊗ (K)

то выделенное из текста слово уже ранее встречалось, и нужно прибавить единицу к W%(K) — показателю частоты встречаемости этого слова:

W%(K) = W%(K) + 1

Но если мы попали на строку 110 оттого, что K превысило L, стало равным L + 1, не дождавшись ни одного совпадения W ⊗ с W ⊗ (K), тогда будет выполняться строка 120:

120 L = K : W ⊗ (L) = W ⊗ : W%(L) = 1

Количество слов в словаре будет помечено новым значением L = K, т. е. L = L + 1. Новое слово будет занесено в словарь именно под таким номером и с начальным показателем частоты встречаемости, равным единице.

Читаю следующую строку 130 осторожно, не доверяясь радостному чувству одержанного успеха. Итак, S = E. Заключительный разделительный знак просмотренного слова становится начальным разделительным знаком слова, которое будет просматриваться следующим.

Программа, словно гусеница, ползет по тексту. Если голова гусеницы попадает на пробел или знак препинания, то она подтянет к этому месту хвост (S = E), а голову начнет продвигать дальше (E = E + 1). Если, конечно, не достигнут конец строки.

А если достигнут? IF E = LEN (S ⊗)? Переходим на строку 150 программы, прибавляем единицу к счетному индексу I, который должен дорасти до N. Дорос — значит, все строки стихотворного текста просмотрены. На дисплее надпись: всего в тексте столько-то слов. А далее — словарь. Он печатается циклом: FOR K = 1 TO L. Печатается слово W ⊗ (K), его частота W%(K).

Программу не понять целиком, не уяснив отчетливо смысла каждого из непривычных ее знаков. А смысл отдельного знака не уяснить, не поняв всей программы целиком.

Блез Паскаль в свое время тонко подметил различие между двумя основными разновидностями познания, математическим и непосредственным. «Начала математического познания отчетливы, но в обыденной жизни неупотребительны, поэтому с непривычки в них трудно вникнуть; зато всякому, кто вникнет, они совершенно очевидны . . . Начала непосредственного познания, напротив, распространены и общеупотребительны . . . Этих начал так много и они так разветвлены, что охватить их сразу почти невозможно. Меж тем пропустишь одно — и ошибка неизбежна».

Математически строгая логика представляется иному характернейшей чертой программирования. Внимание к каждой строке программы, к каждому символу, к мельчайшей частности сближает программирование с искусством.

Комментарии в программе

Можно понять радость поэта, разгадавшего смысл таинственной переменной L. Однако тот, для кого программирование — не экзотическое развлечение, а повседневная работа, скорее всего испытал бы в подобной ситуации не радость от найденной разгадки, а сожаление о времени, затраченном на ее поиски. Чтобы избежать этой невосполнимой траты, достаточно было бы поставить в программе лишнюю ремарку.

Пусть помнит об этом тот, кто желает, чтобы написанные им программы были удобны в работе.

Еще один оператор ввода

Призыв быть внимательным к каждому знаку программы неоднократно звучал из уст поэта, чей монолог помещен на этих страницах. Нельзя сказать, однако, что сам поэт всегда следует своему призыву. Попробуем подметить то, что просмотрел он.

Заметили ли вы, читатель, что имя переменной W%, выражающей частоту встречаемости слов в тексте, помечено знаком процента? Он указывает, что эта переменная может принимать лишь целочисленные значения. Транслятор учитывает это и отводит под хранение целочисленной переменной меньший объем памяти, чем для хранения вещественной переменной, т. е. такой, которая может принять также и дробное значение.

Обратили ли вы внимание, что в строке 30 употреблен не совсем обычный оператор ввода — не INPUT, а LINPUT? Что это — опечатка? Нет, это новый оператор. Он не только запрашивает у пользователя значение указанной в нем переменной, но и выводит на дисплей ее прежнее значение. Теперь, если нажать лишь клавишу ВК, оно сохранится. В текст, который появился на дисплее, можно внести поправки — и тогда после нажатия клавиши ВК переменная получит новое значение.

Еще одно отличие: оператор INPUT, встретив во вводимом тексте запятую, воспринимает ее как разделитель между значениями различных переменных. Оператор LINPUT свободен от такого «предубеждения»: набираемый на клавиатуре текст он вводит с точностью до каждого знака.

Применения частотного словаря

У программы «Частотный словарь» есть небольшой изъян, а может быть, и большой — как считать: ЭВМ одно и то же слово, примененное в разных падежах, принимает как разные. Научить компьютер разбираться в русских падежах, склонениях, спряжениях, родах, наклонениях и прочем не менее сложно, чем обучить его расчетам траекторий искусственных спутников земли.

Грамотному человеку не составит особого труда довести частотный словарь до конца, объединив однокоренные слова. Нужно лишь расположить их в алфавитном порядке. (Вот где пригодится решение задачи о расстановке русских слов по алфавиту, предложенной несколькими страницами ранее.)

Составить частотный словарь бывает полезно не только для стилистического анализа текста. Такую операцию нелишне провести перед окончательной распечаткой введенного в ЭВМ текста. Если при этом окажется, что слово «компьютер», например, использовалось 10 раз, а слово «компьюРер» — только один, ясно, что была допущена опечатка, которую нужно исправить. Но если при этом в тексте всего сто слов, то десятикратное употребление слова «компьютер» говорит о скудости словарного запаса автора. Слишком часто повторяющемуся слову нужно подобрать замену из числа его синонимов — ЭВМ, машина и др.

Этюд 19. ПЕРВАЯ ВСТРЕЧА С ФАЙЛАМИ

Современную молодежь уже ничем не удивишь! Все те сногсшибательные возможности вычислительной техники, которые когда-то так поражали нас, кажутся нашим детям чем-то совершенно естественным, само собой разумеющимся.

— Ты хотел бы еще разок сыграть в игру «Животные»? — спросил я сына, когда мы снова пришли к моему приятелю: я — в гости, а сын — на очередное занятие по информатике, которое, как всегда, было облечено в форму игры.

Сын не стал возражать, и мой приятель включил машину.

«Это кот?» — задал компьютер свой традиционный первый вопрос. «Нет», — ответил сын, нажав клавишу «0». «Живет оно в воде?» — «Нет». — «На нем можно ездить?» — «Да». — «Это слон?» — «Нет». — «Может оно скакать?» — «Да». — «Это лошадь?» — «Нет». — «У него есть горб?» — «Нет» . . .

— Послушай, — нетерпеливо произнес мой приятель, — ты играешь уже добрых пятнадцать минут. Разве ты не заметил, что сегодня эта игра идет не совсем так, как в тот день, когда ты играл в нее первый раз? Неужели не заметил?! Вспомни, как часто компьютер тогда сдавался, просил назвать, кого ты загадал, а потом еще просил сформулировать различие между животными, которое загадал ты и которое имел в виду он перед последним вопросом. Сегодня он играет с тобой так, как будто помнит все, что было тогда, все свои вопросы и все твои ответы. Но ты ведь видел, как я включал машину,

Рис. 4.1. Магнитный диск (в конверте)

и ты прекрасно знаешь, что у только что включенной ЭВМ оперативная память совершенно пуста́.

Сын пожал плечами:

— Наверное, у машины есть еще какая-то другая память.

Мой приятель обезоруженно замолчал. Столь естественной и точной реплики он не предвидел. Он протянул руку к стоявшему рядом с компьютером невысокому ящичку с двумя горизонтальными прорезями и достал из одной тонкий черный квадрат размером с почтовую карточку.

— Ты прав, — сказал он. — Персональный компьютер может иметь запоминающие устройства двух видов. Одно всегда находится в корпусе машины. Оно позволяет быстро считывать информацию, необходимую для выполнения указанных в программе операций, и быстро записывать результаты. Поэтому его и называют *внутренней, оперативной памятью*. Но эти запоминающие устройства очень дороги. Есть, однако, и такие, которые не отличаются расторопностью при записи и чтении информации, зато гораздо дешевле, да и вместительнее. Их подсоединяют к машине так, чтобы в нужный момент их содержимое переписывалось во внутреннюю память и использовалось оперативно. Называют их тоже по-разному: *внешняя память, долговременная память, накопитель*.

— Вот как выглядит накопитель информации у моего компьютера, — он протянул сыну черный квадрат, который держал в руке. Здесь внутри — *магнитный диск* (рис. 4. 1). На каждой его стороне можно четыре раза записать все содержимое оперативной памяти машины. 256 *килобайт* — на маленьком тоненьком кружочке! Если бы ты захотел ввести это с клавиатуры, ты потратил бы несколько суток! С диска же машина проглатывает такую уйму информации за считанные секунды! А если нужно считать только какую-то определенную порцию

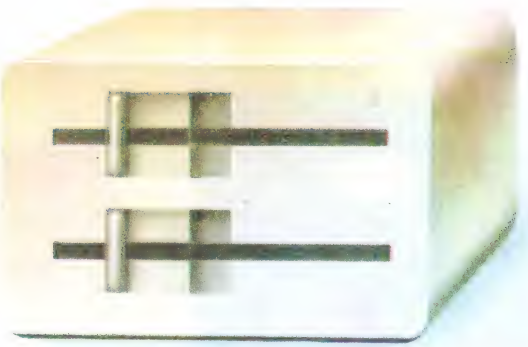


Рис. 4.2. Дисковод

информации — ЭВМ отыщет ее на диске моментально и безошибочно! И так же быстро и точно запишет ее на любой участок диска, который ты укажешь!

Сын невозмутимо пропустил мимо ушей эти красноречивые сведения:

— Скажите: как пользоваться этим диском?

Моему другу окончательно стало ясно, что желаемого ошеломляющего эффекта тут не произведешь, так что не остается ничего другого, как приступить к запланированному уроку. Приняв спокойный деловой тон, он продолжал:

— Конечно, вряд ли когда возникнет необходимость переписывать на диск всю память компьютера. Переписывается, как правило, лишь определенный упорядоченный набор данных. Такой набор, который рассматривается как одно целое и так целиком либо заносится на магнитный диск, либо считывается с него, — такой набор называется *файлом*. Каждый файл имеет свое имя. В качестве имени может служить любая литерная константа. Считывание и запись производятся с помощью дисковода (рис. 4. 2), — мой приятель указал на ящик с прорезями и поднес руку к одной из них. — В один из этих карманов вставляется диск. Если он вставлен в верхний карман, то в программе обозначается буквой R (replaceble — сменный), если в нижний — буквой F (fixed — фиксированный). В программе он может обозначаться еще и буквой T (two — два, оба); тогда компьютер при записи или считывании сначала будет искать диск в верхнем кармане, а если не найдет его там — то в нижнем. Чтобы компьютер мог считывать информацию с диска, надо позаботиться лишь об одном — включить дисковод. Чтобы записывать информацию, надо еще снять с диска *защиту от записи*, — он показал на кнопку под горящей красной лампочкой на

```
1 DIM O¤(500),B¤45,P¤1,A¤1:ON ERROR E¤,N¤ GOTO 10
2 I,J=0:O¤="КОТ":GOTO 7
3 J=J+1:IF I>O THEN 4:I=1:GOTO 5
4 I=2*I:IF P¤=A¤ THEN 5:I=I+1
5 CONVERT I TO I¤,(########):
  DATA LOAD DC OPEN RI¤:DATA LOAD DC B¤,O¤,P¤
6 O¤(J)=O¤:PRINT B¤;:INPUT A¤:IF A¤<>P¤ THEN 3
7 PRINT "ЭТО -";O¤;
8 INPUT " УГАДАЛА (1-ДА,0-НЕТ) ",P:IF P=0 THEN 3
9 INPUT "ЕЩЕ БУДЕТЕ ИГРАТЬ ",P:IF P=1 THEN 2:STOP
10 INPUT "СДАЮСЬ. КТО ЭТО ",O1¤
11 FOR K=1 TO J:IF O1¤<>O¤(K) THEN 12:PRINT
   "НЕЧЕСТНО ИГРАЕТЕ.":GOTO 9
12 NEXT K
13 PRINT "ЗАДАЙТЕ АЛЬТЕРНАТИВНЫЙ ВОПРОС,ОТВЕТИВ НА
   КОТОРЫЙ МОЖНО ОТЛИЧИТЬ ";O1¤;" ОТ ";O¤;:INPUT B¤:
   INPUT "ПРАВИЛЬНЫЙ ОТВЕТ ",P¤:DATA SAVE DC OPEN R(2)
   I¤:DATA SAVE DC B¤,O1¤,P¤:DATA SAVE DC END:GOTO 9
```

Рис. 4.3. Программа «Животные 2»

корпусе дисковода. — Эта дополнительная предосторожность устроена затем, чтобы не допускать до записи непосвященных. Они, чего доброго, выполнят ее так, что потом нечего будет считывать.

Мой приятель снова взял в руки черный квадрат:

— На каждой стороне такого диска — тысяча секторов емкостью по 256 байт каждый. Перед записью информации ты должен указать, сколько секторов выделяешь под запись. Если записываешь или считываешь несколько величин, укажи их в операторе записи или считывания через запятую. С массивами проще: указать надо лишь имя массива, а потом проставить пустые скобки. Тогда записываться или считываться будут все элементы массива подряд. Конечно, в подобных случаях можно указывать и несколько массивов, тоже через запятую — тогда при записи или считывании будут перебираться все элементы первого массива, потом второго и т. д.

Отложив диск, мой друг вынул из стола несколько листингов:

— Операторы считывания и записи файлов имеют в Бейсике довольно пространную длину. В 5-й строке программы, по которой машина играла с тобой сегодня, — мой приятель нашел это место в первом из листингов (рис.4.3), — есть такая запись: DATA LOAD DC OPEN R I ⊗.

Давай сначала разберем отдельные слова. LOAD — значит загрузить, а на профессиональном языке программистов еще и считывать. OPEN — открыть. А все вместе означает: на диске R открыть для считывания файл под именем, выражаемым

литерной переменной I ⊗. Далее в программе стоит собственно оператор считывания: DATA LOAD DC B ⊗, O ⊗, P ⊗. Три считанных с диска значения будут присвоены трем литерным переменным, указанным в этом операторе через запятую. А вот запись, которая помещается в самой последней, 13-й строке программы: DATA SAVE DC OPEN R(2)I ⊗. Разберем ее. SAVE — значит хранить, а на языке программистов — записывать. Все вместе это означает: на диске R открыть 2 сектора для записи файла под именем, выражаемым литерной переменной I ⊗. Читаем далее. DATA SAVE DC B ⊗, ○1 ⊗, P ⊗ — перечисленные через запятую переменные записываются на указанное место. DATA SAVE DC END — запись окончена. Записанное можно стереть оператором SCRATCH, — перебрав лежащие на столе листинги, приятель нашел такой оператор в одной из программ. — После этого служебного слова, которое так и переводится — стереть, указывается диск и имя ликвидируемого файла. Место, прежде занятое этим файлом, останется свободным, и туда можно записать другую информацию, которая будет значиться на диске под тем же именем, если ты, конечно, не дашь при записи этому файлу другое имя.

Мой друг сделал паузу:

— Тебе все понятно? — спросил он сына. Тот кивнул. — Что ж, я рад за тебя. Хотя предупреждаю: ошибки при считывании и записи файлов случаются даже у довольно опытных программистов. Например, в программе указывается, считать такой-то файл, а этого файла на диске нет. На такие случаи в Бейсике есть специальный оператор. Вот взгляни, — он указал в конец 1-й строки: ON ERROR E ⊗, N ⊗ GOTO 10. Если произойдет ошибка, переменная N ⊗ станет равной номеру строки, где машина совершит неверное действие, а переменная E ⊗ станет равной коду ошибки. Например, попытка считывания несуществующего файла кодируется числом 73. И какая бы ошибка ни произошла, управление будет передано на строку, указанную после оператора ON ERROR, в нашей программе — на строку 10. Кстати, *переходы по программной ошибке* можно отменять — для этого в программе надо поставить оператор ON ERROR без единого символа за ним.

Сын критическим взглядом изучал программу:

— Код ошибки — число, номер строки — тоже число. А оба записываются как литерная переменная — E ⊗, N ⊗.

— Так принято в данной версии Бейсика, — пожал плечами мой друг. — Но трудностей это никаких не создает. Ты в первой же строке указываешь формат записи обеих переменных, скажем, четыре байта, указываешь вот так: E ⊗ 4, N ⊗ 4, и при попытке считывания несуществующего файла происходит присваивание E ⊗ = «0073»; так же записывается и номер строки

N ⊗, где произошла ошибка. Да нам в этой программе ни то, ни другое число и не понадобятся, потому и нет в ней никаких указаний о формате переменных E ⊗ и N ⊗. Сигналом к переходу на строку 10 будет сам факт ошибки.

Он вдруг осекся:

— Что это я — сразу про десятую строку? Давай по порядку, с начала. Впрочем, начальные строки вряд ли нуждаются в пояснениях. Строка 2, далее строки 7, 8 и 9, куда мы переходим после строки 2, — все они такие же, как в прежней программе, по которой машина играла с тобой в первый раз* (см. рис. 3.1). Почти такие же строки 3 и 4. Разночтения начинаются в строке 5, когда машина должна выдать на дисплей свою догадку. Число I, номер ветки, до которой ЭВМ добралась, она превращает в литерную переменную I ⊗. Под таким именем на диске записан файл, хранящий информацию о животном, «висящем» на той или иной I-й ветке, а дальше — оператор, открывающий этот файл для считывания, знакомый тебе оператор DATA LOAD DC OPEN R I ⊗, затем — оператор считывания, тоже тебе знакомый: DATA LOAD DC B ⊗, O ⊗, P ⊗. Три литерные переменные, считанные при его выполнении, — это альтернативный вопрос B ⊗, догадка машины O ⊗ и признак правильности ответа на вопрос P ⊗. Машина дублирует свою догадку в переменной O ⊗ (J), выводит на дисплей вопрос B ⊗, запоминает твой ответ A ⊗, сравнивает его с признаком P ⊗, чтобы в случае их несовпадения вернуться на строку 3 и продолжить диалог с тобой. Но если они совпали, то весьма вероятно, что ты загадал то же самое животное, которое имеет в виду машина. Она спросит тебя об этом в строке 7. Диалог продолжится, если ты ответишь «нет». При каком-то из твоих отрицательных ответов может получиться так, что у машины нет в запасе никакой догадки, т. е. что на ветке под номером I ничего не висит. А если говорить без иносказаний — файла с именем I ⊗ на диске нет.

— Тогда второй оператор строки 5 не выполнится! — нетерпеливо вставил сын.

— Да, но тогда выполнятся операторы строки 10, — тотчас подхватил мой друг. — Попытка считать несуществующий файл — это ошибка. А на случай программной ошибки мы еще в первой строке написали: ON ERROR E ⊗, N ⊗ GOTO 10, т. е. в случае ошибки перейти на строку 10. Что там? Машина сдается, просит тебя раскрыть ей задуманное тобой животное и запоминает его в виде литерной переменной O1 ⊗. В стро-

---

* При сравнении программ следует учесть, что во всех программах этой главы строки нумеруются последовательными целыми числами и переменные не снабжаются пояснениями. Надеемся, что читатели уже приобрели достаточный опыт разбора программ, чтобы согласиться с такими упрощениями.

к а х 11 — 12 машина проверит, не было ли его среди названных ею прежде. В с т р о к е 13 попросит сформулировать альтернативный вопрос В ⊗, отличающий загаданное тобой животное от ее последнего предположения, потом попросит сообщить ей правильный ответ Р ⊗ на этот вопрос. И тогда она начнет записывать информацию, которую ты ей сообщил, в виде файла под именем 1 ⊗ — под именем того же вида, что и сведения обо всех знакомых ей раньше животных. Теперь файл 1 ⊗, отсутствие которого привело машину к ошибке при попытке его считать, будет создан. Для этого машина откроет на диске R два сектора для записи: DATA SAVE DC OPEN R(2) 1 ⊗. Далее ЭВМ запишет все те сведения, которые должны иметься для каждого животного, — альтернативный вопрос, название, признак правильного ответа: DATA SAVE DC B ⊗, O1 ⊗, P ⊗, а по окончании записи закроет новообразованный файл. После этого она перейдет на строку 9 и спросит тебя: БУДЕТЕ ЕЩЕ ИГРАТЬ?

— Будем! — улыбнулся сын своему учителю.

— Но все-таки ответь мне на один вопрос, — с какой-то умоляющей ноткой в голосе произнес мой приятель. — Как ты догадался, что у машины есть еще какая-то память, кроме оперативной?

Сын не стал лукавить:

— Когда вы только начинали рассказывать мне о программировании, вы каждую программу вводили с пульта. А потом просто вставляли в дисковод диск, нажимали совсем немного клавиш, и программа была уже в машине.

— Ну, конечно! — хлопнул себя по лбу мой приятель. — Для меня это так естественно, что я этого и не замечаю. Ну что ж — тогда продолжим наши занятия! То есть я хотел сказать — продолжим наши игры!

Запись файлов на диск

На магнитный диск данные записываются примерно так же, как в обычную тетрадь заносится любая запись. Тетрадь для этого нужно открыть, написать на чистом листе название записи, по которой ее впоследствии можно будет найти, записать нужную информацию и закрыть тетрадь. Аккуратный человек отводит первый лист тетради под каталог, где перечисляются названия имеющихся в ней записей с указанием страниц, где они находятся.

Читатель, следивший, как протекает игра «Животные» с компьютером, оборудованным дисководом, уже получил представление о том, как ведется считывание и запись файлов на одной из версий Бейсика. Это была версия машины «Искра 226». Посмотрим теперь, как производятся те же действия на машинах ДВК-2М и СМ-4.

```
1 OPEN «MASA» FOR  OUTPUT AS FILE # 1
2 FOR I = 1 TO N : PRINT # 1,A(I):NEXT I
3 CLOSE # 1
```

В этом фрагменте операторы строки 1 открывают файл с именем «MASA» на диске 1 для записи значений элементов массива A( ), операторы строки 2 записывают в этот файл значения этих элементов, операторы строки 3 файл закрывают.

Впоследствии записанные на диск данные можно считать так:

```
4 OPEN «MASA» FOR INPUT AS FILE # 1
5 FOR I = 1 TO N:INPUT # 1,A(I):NEXT I
```

Операторы строки 4 открывают для считывания файл с именем «MASA» на диске 1, операторы строки 5 считывают данные и присваивают их соответствующим элементам массива A( ). После чтения файл, как правило, можно не закрывать.

В этой книге описывается только один из способов организации файлов, при котором возможен лишь последовательный доступ к записанным в них данным: чтобы считать, например, значение пятого элемента массива, нужно перед этим считать значения четырех первых.

На магнитном диске можно организовать файлы с непосредственным доступом к отдельным данным. Для этого они записываются на диск в виде так называемых *виртуальных массивов*.

Разбиение диска

Поверхность магнитного диска разбита на зону *каталога*, где записаны заголовки хранящихся на диске файлов, их длина и координаты начала, и на зону *хранения самих файлов*, которая в свою очередь делится на дорожки и сектора. Такую разметку (инициализацию) диска необходимо провести перед первым его использованием для записи программ или данных. Деление диска при этом может быть статическим, например как на ЭВМ «Искра 226», или динамическим, как на ДВК-2. При динамической разбивке диска машина в процессе работы сама может увеличивать зону каталога за счет зоны файлов, и наоборот. При статической разбивке программист должен заранее предусматривать, что у него будет храниться на диске, и при необходимости расширять зону каталога, если на диске предполагается записывать большое количество коротких файлов (например, как во втором варианте игры «Животные» — см. рис. 4. 3).

Если компьютер получит команду перенести информацию с диска в оперативную память, то он по имени требуемой записи найдет в каталоге координаты ее начала, повернет диск на нужный угол, передвинет магнитную головку на нужную дорожку и считает информацию. При записи на диск информация помещается на незанятые сектора, координаты которых вместе с именем записи заносятся в каталог.

Этюд 20. ПАМЯТЬ НАПРОКАТ

Когда я был студентом, больше всего хлопот мне доставлял иностранный язык. Память моя устроена так, что мне легче запомнить дюжину математических выкладок, чем перевод одного английского слова. Подготовка и сдача преподавателю пресловутых «тысяч» — статей из газеты «Moscow news» — для меня были сущей мукой. Мой англо-русский словарь был совсем истрепан. Переводя текст, я лазил в словарь за одним и тем же словом по нескольку раз. Записать же перевод я считал лишним, будучи уверен, что теперь-то уж запомнил его на всю жизнь. Но это было очередной иллюзией, и через пять минут я снова искал в словаре то же самое слово.

Я и сейчас, когда работаю над статьей или штудирую новую книгу, частенько беру с полки словарь — орфографический или политехнический, синонимов или иностранных слов. Но пользуюсь им совсем не так, как в студенческие годы. Теперь у меня есть персональный компьютер с дисководом. Я превратил его в электронный словарь-справочник с удобнейшими средствами поиска, исправления и расширения информации. Хранится она на магнитном диске, а выдается в таком вот виде:

MEMORY (АНГЛ) — 1) ПАМЯТЬ, ЗАПОМИНАНИЕ,
2) ВОСПОМИНАНИЕ, 3) ЗАПОМИНАЮЩЕЕ УСТРОЙСТВО.
ЭВМ — КОМПЬЮТЕР, ВЫЧИСЛИТЕЛЬНАЯ МАШИНА, КАЛЬКУЛЯТОР,
COMPUTER (АНГЛ.), RECHNER (НЕМ.), ORDINATEUR (ФР.).
МИЛЯ — ГЕОГРАФИЧЕСКАЯ — 7420 М, МОРСКАЯ — 1852 М.
ФАЙЛ, FILE — НАБОР СЕМАНТИЧЕСКИ СВЯЗАННЫХ МЕЖДУ СОБОЙ ДАННЫХ, ВОСПРИНИМАЕМЫХ КАК ОДНО ЦЕЛОЕ, ХРАНЯЩИХСЯ В ФОРМЕ МАГНИТНЫХ ЗАПИСЕЙ НА ДИСКЕ.

```
1 DIM P⊠250,P1⊠250,E⊠4,N⊠4:ON ERROR E⊠,N⊠ GOTO 8
2 C⊠=" ":LINPUT"КЛЮЧЕВОЕ СЛОВО ",C⊠:IF C⊠=" " THEN 13
3 DATA LOAD DC OPEN R C⊠:DATA LOAD DC P⊠
4 P1⊠=P⊠:PRINT"ИНФОРМАЦИЯ-";:LINPUT P⊠:IF P1⊠=P⊠ THEN 2
5 SCRATCH R C⊠:DATA SAVE DC OPEN R(C⊠) C⊠:DATA SAVE
  DC P⊠:DATA SAVE DC END:GOTO 2
6 P⊠=" ":LINPUT"ИНФОРМАЦИИ НА ДИСКЕ НЕТ. ДАЙТЕ ЕЕ",P⊠:
  IF P⊠=" " THEN 2
7 DATA SAVE DC OPEN R(2) C⊠:DATA SAVE DC P⊠:
  DATA SAVE DC END:GOTO 2
8 IF N⊠="0003" AND E⊠="73" THEN 6:REM НЕТ ФАЙЛА
9 IF N⊠="0003" THEN 11:REM ДИСКОВОД НЕ В ПОРЯДКЕ
10 IF E⊠="8005" THEN 12:REM С ДИСКА НЕ СНЯТА ЗАЩИТА
11 INPUT"ЧТО-ТО НЕ В ПОРЯДКЕ С ДИСКОВОДОМ.ИСПРАВИЛИ",A⊠
   :GOTO 3
12 INPUT"СНИМИТЕ ЗАЩИТУ С ДИСКА.ГОТОВО",A⊠:
   IF N⊠="0005" THEN 5:GOTO 7
13 ON ERROR :REM ВЫКЛЮЧЕНИЕ АНАЛИЗАТОРА ОШИБОК
```

Рис. 4.4. Программа «Энциклопедический словарь»

Встретив незнакомое слово, я первым делом запрашиваю сведения о нем у своего электронного референта. Если ему есть что сообщить в ответ, он выводит информацию с диска на дисплей. Если же нужной мне информации у него нет, он запрашивает ее у меня. Вот тогда-то я беру с полки словарь и возвращаюсь с ним к компьютеру. Найдя в словаре толкование незнакомого термина, я записываю его на магнитный диск. Точно так же поступаю я, если ответ электронного референта меня не удовлетворяет. Выведенную на дисплей информацию я исправляю или дополняю, а потом записываю заново на тот же участок диска, где она помещалась раньше.

Любопытно, что программа (рис. 4.4), по которой идет вся эта непростая работа с машиной, ни в одной своей строке не содержит ни одной числовой переменной, ни одной вычислительной операции. Зато почти в каждой строке есть нечто интересное с программистской точки зрения.

Строка 1. Первая из описываемых тут литерных переменных, P ⊠, имеет вполне понятный смысл: это справка, выдаваемая машиной по введенному в нее с клавиатуры ключевому слову C ⊠. Справка может содержать не более 250 знаков, т. е. занимать не более одного сектора магнитного диска. Переменная P1 ⊠ дублирует эту справку на случай возможных исправлений. А две следующие переменные, каждая объемом в четыре байта, нужны для работы так называемого *анализатора ошибок*. Тот, кто знаком с таким удобным средством программирования, конечно, уже заметил в конце строки оператор, запускающий этот анализатор: ON ERROR E N ⊠ GOTO 8.

Если при выполнении программы возникнет ошибка, то ее код станет значением литерной переменной E ⊗, а номер строки, где произошла ошибка, — значением переменной N ⊗. Четырех байт для выражения такой информации вполне достаточно. Управление программой при этом будет передано на строку 8, где машина станет решать, как ей в такой ситуации поступать.

Строка 2. C ⊗ = « ». Переменная C ⊗, которая будет хранить ключевое слово, очищается не случайно. Если вы нажмете клавишу ВК, ничего не набрав на клавиатуре, и переменная C ⊗ остается пустой, то машина воспримет это как сигнал к прекращению диалога IF C⊗ = « » THEN 13. Управление будет передано на строку 13, выключающую анализатор ошибок. Так как она последняя в программе, то после ее выполнения компьютер прекратит работу.

Строка 3. Тут все ясно. DATA LOAD DC OPEN R C ⊗: на диске R машина открывает для считывания файл, именем которого является ключевое слово C ⊗. DATA LOAD DC P ⊗: считанное содержимое файла машина присваивает литерной переменной P ⊗.

Строка 4. Найденную справку P ⊗ компьютер дублирует в переменной P1 ⊗ и выводит на дисплей, предварив примечанием ИНФОРМАЦИЯ. И вот что интересно: примечание выводится обычным способом, с помощью оператора PRINT, а сама справка — не совсем обычным, с помощью оператора LINPUT. Тут он очень удобен: он и переменную P ⊗ на дисплей выведет, и выполнение программы приостановит, давая возможность прочесть текст на дисплее и исправить его при необходимости. Прочли — нажимайте клавишу ВК. Если вы что-то изменили в выведенной справке, то компьютер перепишет ее на диск в исправленном виде, перейдя на строку 5. Если ничего не изменили — будет ждать ввода нового ключевого слова, перейдя на строку 2. Такое ветвление осуществляется стоящим в конце строки условным оператором IF r ⊗ = P1 ⊗ THEN 2. Направление перехода определяется тем, совпало ли прежнее значение переменной P ⊗, продублированное в P1 ⊗, с новым — тем, что она имеет к моменту нажатия клавиши ВК.

Строка 5. Попадаем сюда, если справка по ключевому слову должна быть заменена новой. SCRATCH R C ⊗: ликвидируем файл с именем C ⊗ на диске R; DATA SAVE DC OPEN R (C ⊗) C ⊗: открываем с тем же именем на том же месте файл для записи. DATA SAVE DC P ⊗: записываем в него значение переменной P ⊗. DATA SAVE DC END:GOTO 2: запись окончена, идем на строку 2 за новым ключевым словом.

Непосвященного, быть может, удивит, что в операторе записи DATA SAVE DC OPEN R (C ⊗) C ⊗ в скобках стоит не число, указывающее количество отводимых под запись секто-

ров диска, а литерная переменная. А между тем нетрудно догадаться, что тут имеется в виду: новый файл С ⊗ запишется в те же самые сектора, которые до этого занимал старый, имевший то же имя, и это имя за ним сохранится.

Строка 6, загадочная для того, кто читает программу впервые и к тому же не очень сведущ в тонкостях работы с анализатором ошибок. Для него может остаться непонятным: как машина попадает на эту строку? Откуда? И почему? Этот переход начинается в строке 3, в операторе DATA LOAD DC OPEN R C ⊗. Выполняя его, машина ищет на диске R файл под именем С ⊗. Если же она его не находит, анализатор ошибок передает управление на строку 8. Тут и начинается сеть логических путей, один из которых ведет на строку 6. Эту сеть стоит просмотреть всю целиком.

Строка 8. IF N ⊗ = «0003» AND E ⊗ = «0073» THEN 6. Оператор выполняется, если ошибка произошла в строке 3 и код ошибки равен 73, т. е. произошла она потому, что на диске отсутствует файл с именем С ⊗. Вот тогда-то мы и переходим на строку 6 и можем прочесть ее заново.

Строка 6. Тот, кто разобрался в строке 2, поймет, что на сей раз компьютер поступает точно таким же образом: запрашивает значение переменной Р ⊗ и присваивает ей тот текст, который будет набран на клавиатуре. Если же не будет набрано ничего (не нашел человек справку в книгах), компьютер возвратится на строку 2, ожидая новое ключевое слово: IF P ⊗ — = « » THEN 2.

Строка 7. На ней мы оказываемся, если переменная Р ⊗ не пуста. Тут все ясно тому, кто разобрался в строке 5: на диске R открываются два сектора для записи нового файла под именем С ⊗, производится запись, файл закрывается, управление передается на строку 2.

Строка 9. Анализ ошибок продолжается. На эту строку машина переходит, если в предыдущей не выполнилось одно из предположений N ⊗ = «0003» AND E ⊗ = «0073». Если все-таки ошибка произошла в строке 3, а причина ее не в том, что на диске не нашлось искомого файла — значит, что-то не в порядке с дисководом. Перейдя на строку 11, машина прямо заявит об этом.

Строка 11. Но выведя свое заявление на дисплей, машина напечатает далее: ИСПРАВИЛИ? В программе знака вопроса нет, но тот, кто знаком с работой оператора INPUT, знает, что он всегда ставит вопросительный знак, закончив вывод записанного в нем текста, если вслед за текстом в кавычках стоит запятая. Этот оператор требует, чтобы вы что-нибудь ввели с клавиатуры. Нажмите любую клавишу или ничего не делайте — значение, которое будет присвоено переменной А ⊗, никакой роли не играет. Переменная А ⊗ нигде в программе не исполь-

зуется; программисты в таких случаях называют переменную пустой. Здесь важно лишь то, что оператор INPUT приостанавливает выполнение программы. Машина подождет, пока вы предпримете необходимые исправления и сообщите ей об этом нажатием клавиши ВК. После этого она повторит поиск нужной вам информации на диске, совершив переход GOTO 3.

Строка 10. Продолжение анализа ошибки. На эту строку приходим, если ошибка произошла не в строке 3, т. е. ее причина не в неудавшемся считывании с диска. Может быть, не удается запись на диск? Может быть, с него не снята защита? Код такой ошибки как раз равен 8005.

Строка 12. Машина ждет, пока вы снимете защиту с диска. ГОТОВО? Переменная A ⊗ — опять-таки пустая. Ошибка, которая привела на эту строку, могла возникнуть либо в строке 5, либо в строке 7. Машина разберется в этом и вернется туда, откуда пришла, чтобы повторить ввод: IF N ⊗ = «0005» THEN 5:GOTO 7.

Строка 13. Окончание работы по программе.

Я доволен своим электронным словарем. На собственную память у меня по-прежнему надежда слабая. Но на магнитную, как я убедился, положиться можно вполне.

Банки данных

Банк данных. Без этих слов невозможно понять ту роль, которую играют компьютеры в современной жизни. Располагая банком данных, машина способна моментально дать запрошенную у нее справку, способна стать мудрым советчиком человека. ЭВМ, не имеющая банка данных, подобна человеку, лишенному памяти.

Вся эта глава посвящена организации банков данных, хотя сам термин ни разу в ней не произносится. Без него можно обойтись, потому что это понятие здесь предстает в образе хорошо знакомых всем предметов: записной книжки, тетради с заучиваемыми словами иностранного языка, сборника кулинарных рецептов, списка телефонов, лежащего под стеклом письменного стола, . . .

Каждое из этих средств мы ценим за возможность расширения и исправления записей, легкость и быстроту поиска информации. Читая эту главу, можно убедиться, что банк данных на магнитном диске намного лучше отвечает таким требованиям, чем его бумажные аналоги.

Операции с файлами

В Бейсике есть немало операторов для работы с файлами. Файлы можно ликвидировать, переименовывать, перегружать с диска на диск, распечатывать на принтере и т. д. Освоить эти операторы можно, лишь сидя за дисплеем с руководством по программированию в руках. Поэтому мы и не рассказываем про них.

Этюд 21. ПЯТИМИНУТКИ С «ИСКРОЙ»

Наш завлаб Николай Петрович Орлов (за глаза мы зовем его НП) — хороший ученый, отличный человек, но администратор — не очень. Как и всякому начальнику, ему приходится отдавать разные распоряжения своим подчиненным. С этой задачей НП справляется неплохо. А вот что касается проверки исполнения, то здесь у него не ладится.

Когда НП узнал, как электронно-вычислительная техника облегчает труд руководителя, он вызвал меня, поскольку ЭВМ «Искра 226» находится в моем ведении, и поручил составить электронную картотеку приказов по лаборатории. НП высказал свои пожелания, перечислил необходимые виды работы с картотекой:

1. Записать на первую свободную карточку новое распоряжение с указанием дат выдачи и выполнения, имени исполнителя и начальника, отдавшего приказ, а также номера папки, хранящей дополнительную информацию.
2. Снять с контроля выполненное распоряжение.
3. Узнать, что нужно сделать к текущему дню.
4. Получить информацию о просроченных распоряжениях и о тех, срок исполнения которых приближается.
5. Просмотреть всю картотеку.
6. Просмотреть отдельные папки с дополнительной информацией.
7. Просмотреть карточки по отдельным исполнителям и начальникам.

Карточка должна иметь вид примерно такой, как на рис. 4.5.

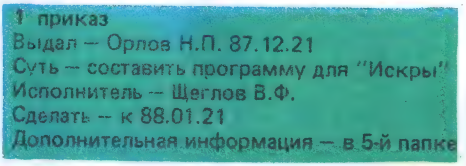


Рис. 4.5. Карточка из картотеки приказов

В тот же день я сочинил небольшую программу на Бейсике, а на следующее утро представил ее НП.

В свое время, когда НП приказал всем сотрудникам нашей лаборатории пройти курс лекций по Бейсику, он и сам аккуратно посещал их. Потом, когда для любой задачи, решаемой кем-нибудь из нас, он велел составлять программу на Бейсике, то внимательно разбирал ее вместе с составителем. Так что и теоретический, и практический курс языка он прошел и в Бейсике разбирается неплохо.

Зная о вечной занятости своего шефа, я постарался составить программу из коротких блоков, чтобы в ней легко было разобраться.

НП взял листинг программы (рис. 4.6) с напечатанным списком переменных и стал читать, а я, следя за его взглядом, пояснял, пропуская очевидные моменты и задерживаясь на более или менее принципиальных:

— Строка 1. В картотеке сто карточек. Диск и память машины могут хранить и больше, но для нашей лаборатории достаточно ста. Строка 2. Считывание с диска R файла «КАРТОЧКИ» — таким именем я озаглавил картотеку — и сообщение о «свежести» записей, т. е. выдача даты последней правки. Строка 3. Обращение к подпрограмме в строке 5001*, где машина запрашивает дату и определяет ее номер на хронологической оси двадцатого века. Потом проверка: не ввел ли пользователь дату D ⊗, более раннюю, чем дата последней правки D0 ⊗. Тогда запрос повторяется. Строка 4. Меню.

— Меню? Ну давай попробуем все блюда, — пошутил НП. — Вот номер 1. Новое распоряжение.

Я показал в конец меню:

— Переходим на строку 7. Машина ищет пустую карточку, перебирая их все в цикле. Если не нашла, сообщает, что картотека переполнена и возвращается на строку с меню. Если нашла, переходит на следующую строку 8. Здесь печатает номер карточки, заполняет сегодняшним числом D0 ⊗ дату

* Эта подпрограмма — точно та же, что и употребленная в программе «Вечный календарь» (см. рис. 3.6).

```
1 N=100:DIM A¤(2,100),D¤(2,100)8,D(100),S¤(100)100,
  T(100),D¤8,D0¤8:REM ОПРЕДЕЛЕНИЕ РАЗМЕРА МАССИВОВ
2 DATA LOAD DC OPEN R"КАРТОЧКИ":
  DATA LOAD DC A¤(),D¤(),D(),S¤(),T(),D0¤:
  R=0:PRINT "ДАТА ПОСЛЕДНЕЙ ПРАВКИ КАРТОТЕКИ";D0¤
3 PRINT "КАКОЕ СЕГОДНЯ ЧИСЛО";:GOSUB 5001:
  IF D¤<D0¤ THEN 3:D0¤=D¤:D0=D
4 INPUT "1-НОВОЕ РАСПОРЯЖЕНИЕ,2-СНЯТИЕ С КОНТРОЛЯ,
  3-СРОК ИСПОЛНЕНИЯ СЕГОДНЯ,4-ДО СРОКА ОСТАЛОСЬ МЕНЕЕ..
  ДНЕЙ,5-ПРОСМОТР КАРТОТЕКИ,6-ПАПОК,7-ПО НАЧАЛЬНИКАМ,
  8-ПО ИСПОЛНИТЕЛЯМ,9-КОНЕЦ.",A:ON A GOTO
  7,13,17,15,17,16,14,14,5
5 IF R=0 THEN 6:SCRATCH R"КАРТОЧКИ":
  DATA SAVE DC OPEN R("КАРТОЧКИ")"КАРТОЧКИ":
  DATA SAVE DC A¤(),D¤(),D(),S¤(),T(),D0¤:
  DATA SAVE DC END
6 STOP
7 FOR I=1 TO N:IF S¤(I)=" " THEN 8:NEXT I:
  PRINT "КАРТОТЕКА ПЕРЕПОЛНЕНА.":GOTO 4
8 PRINT I;" -Я КАРТОЧКА.":D¤(1,I)=D0¤
9 PRINT "ДАТА ВЫПОЛНЕНИЯ";:GOSUB 5001:IF D¤<D0¤ THEN 9
10 D¤(2,I)=D¤:D(I)=D:PRINT "СРОК-";D-D0;"ДНЕЙ"
11 INPUT "КТО ВЫДАЛ",A¤(1,I):INPUT "СУТЬ ",S¤(I)
12 INPUT "ИСПОЛНИТЕЛЬ",A¤(2,I):INPUT "НОМЕР ПАПКИ С
  ДОПОЛНИТЕЛЬНОЙ ИНФОРМАЦИЕЙ ",T(I):R=1:GOTO 4
13 INPUT"НОМЕР СНИМАЕМОГО С КОНТРОЛЯ РАСПОРЯЖЕНИЯ ",I:
  S¤(I)=" ":R=1:GOTO 4
14 INPUT "КТО ЭТО",A¤:GOTO 17
15 INPUT "ЧИСЛО ДНЕЙ ",P:GOTO 17
16 INPUT "НОМЕР ПАПКИ",T
17 B=0:FOR I=1TON:IF S¤(I)=" "THEN24
18 ON A-2 GOTO 19,20,22,21:
  IF A¤=A¤(A-6,I) THEN 22:GOTO 24
19 IF D¤(2,I)=D0¤ THEN 22:GOTO 24
20 IF D(I)<=D0+P THEN 22:GOTO 24
21 IF T<>T(I) THEN 24
22 IF B=0 OR B/4>INT(B/4) THEN 23:INPUT"ПРОДОЛЖИТЬ",P¤:
  PRINT HEX(03)
23 PRINT I;" ПРИКАЗ.":PRINT "ВЫДАЛ ";A¤(1,I);D¤(1,I);".
  СУТЬ- ";S¤(I):PRINT "ИСПОЛНИТЕЛЬ- ";A¤(2,I);"СДЕЛАТЬ
  К ";D¤(2,I);". ИНФОРМАЦИЯ В ";T(I);"-Й ПАПКЕ.":B=B+1
24 NEXT I:IF B>0 THEN 4:PRINT "ТАКОГО НЕТ.":GOTO 4
```

Рис. 4.6. Программа «Картотека приказов»

выдачи распоряжения D ⊗(1, I). Идет на строку 9, там переходит на ту же подпрограмму в строке 5001, запрашивает дату выполнения, проверяет, не предшествует ли введенная дата D ⊗ дате выдачи распоряжения D0 ⊗. Если предшествует, то запрос повторяется: GOTO 9.

Строка 10. Введенная дата D ⊗ становится датой выполнения выдаваемого распоряжения D ⊗ (2, I), тут же вычис-

ляется номер этого дня на хронологической оси двадцатого века, подсчитывается, сколько дней есть в запасе у исполнителя распоряжения. Строки 11 и 12. Карточка заполняется по всем позициям. Признак R получает значение 1; зачем — я скажу позже. Новое распоряжение оформлено, возвращаемся на строку 4, где записано меню.

— Отлично! — воскликнул НП, довольный моим четким объяснением. — Следующий пункт, снятие с контроля.

— Это совсем просто, — бесстрастно продолжал я. — Идем на строку 13. Вводим номер снимаемого распоряжения I и очищаем карточку с этим номером: S ⊗ (I) = « ».

— Великолепно! — НП воодушевлялся все более. — Давай пункт третий.

— Срок исполнения — сегодня, — прочел я соответствующую позицию меню. — Это просмотр всех распоряжений, срок исполнения которых истекает в указанный вами день. Этот фрагмент программы посложнее предыдущих.

— Все равно посмотрим! — не желал отступать НП, приободренный предыдущим разговором.

— Идем на строку 17. B — это счетчик просмотренных карточек, он нам еще понадобится. Перебираем их в цикле, с 1-й по N-ю. Если карточка пуста, переходим на строку 24, увеличиваем параметр цикла I на единицу и возвращаемся на второй оператор строки 17 для просмотра новой карточки. Если она не пуста, идем на строку 18 и тут сразу попадаем на разветвление, на оператор выбора ON A — 2 GOTO 19, 20, 22, 21. Если мы работаем по третьему режиму меню, т. е. A = 3, то A — 2 = 1 и согласно оператору выбора идем на строку 19. Если дата выполнения D ⊗ (2,I), стоящая на просматриваемой карточке, совпадает с введенной вами датой D ⊗, идем на строку 22. Тут проверяем счетчик просмотренных карточек B на равенство нулю или кратность четырем. Дело в том, что на дисплее может разместиться всего четыре карточки. И если B кратно четырем, т. е. число B/4 равно своей целой части, значит, дисплей заполнен до отказа. Лишь в том случае, когда дисплей чист (B = 0) или на нем меньше четырех карточек (число B/4 не равно своей целой части), машина выведет новую карточку, перейдя на строку 23. Если же дисплей полон, машина спросит вас: ПРОДОЛЖИТЬ? Нажимаете ВК — и оператором PRINT HEX (03) дисплей будет очищен, а затем, после перехода на строку 23, на нем появится новая карточка. Счетчик карточек увеличится на единицу, машина перейдет на строку 24, а оттуда вернется на строку 17, чтобы продолжить просмотр картотеки. Вернется, если I еще не равно N. Если равно, то при ненулевом B машина возвратится на строку 4, где записано меню. Но если B = 0, то машина перед возвращением выведет на дисплей сообщение,

что в картотеке нет карточки, где дата исполнения совпадает с указанной вами.

— Понятно, — заключил мой рассказ НП заметно потускневшим тоном. — Давай пункт четвертый.

— До срока осталось менее указанного вами количества дней. Идем на строку 15. Тут вводим это самое количество дней Р. Далее идем на строку 17 и начинаем циклический просмотр карточек. При ветвлении в строке 18 направляемся уже на строку 20. Здесь по неравенству D(I) $<=$ D0 $+$ P отбираем интересующие нас карточки и выводим их на дисплей тем же порядком, что и по предыдущему пункту меню.

— Неплохо! — чуть оживился НП. — А пункт пятый?

— Просмотр картотеки. Отличается от предыдущего тем, что при ветвлении в строке 18 идем на строку 22, т. е. минуем все отборочные строки и выводим все карточки подряд.

— Шестой пункт! — голос НП зазвенел, свидетельствуя о его поднимающемся настроении.

— Просмотр папок. Этот режим отличается от предыдущего тем, что на строке 16 вводится номер папки и согласно условному оператору в строке 21 карточка не выводится на дисплей, если она содержится в папке с номером, не совпадающим с введенным.

— Седьмой пункт!

— Отличается тем, что на строке 14 вводится фамилия начальника, отдавшего распоряжение. При ветвлении в строке 18 получается, что A $-$ 2 $=$ 7 $-$ 2 $=$ 5, а пятой позиции в этом операторе выбора нет. Поэтому машина его игнорирует и переходит к следующему оператору, где проверяет на совпадение введенную вами фамилию A $\otimes$ с указанной в карточке фамилией A $\otimes$ (1, I).

— Здесь написано A $\otimes$ (A $-$ 6, I), — остановил меня НП.

— Но поскольку A $=$ 7, то A $-$ 6 $=$ 1 и A $\otimes$ (A $-$ 6, I) $=$ $=$ A $\otimes$ (1, I), — небрежно ответил я.

— Великолепно!

— Отобранную этим условным оператором карточку машина выведет на дисплей прежним порядком, перейдя на строку 22.

— Восьмой пункт! — произнеся это, НП вгляделся в меню. — Восьмой — это проверка по исполнителям. Не посмотреть ли нам, какие распоряжения должен выполнять товарищ Щеглов?

— То есть ваш покорный слуга, — улыбнулся я в ответ. — Посмотрим обязательно, когда наша картотека заработает. Только давайте прежде договоримся о том, как называть исполнителей. Меня можно назвать и тов. Щеглов, и В. Ф. Щеглов, и Щеглов В. Ф., на все я откликнусь. Но для ЭВМ это будут совершенно разные люди. Чтобы машина не сбивалась, надо всех сотрудников — и начальников, и исполнителей — име-

новать единообразно. Скажем, Щеглов В. Ф., Орлов Н. П. Ну, а что касается поиска карточки с названной фамилией, то он выполняется почти так же, как в только что разобранном режиме. Восьмой пункт меню — значит, A = 8, A — 6 = 2, A ⊗ (A — 6, 1) = A ⊗ (2, 1), и машина будет проверять совпадение A ⊗ = A ⊗(2, 1), т. е. совпадение введенной вами фамилии с фамилией исполнителя на карточке.

— Девятый пункт — все, — прочел НП последний пункт меню. — Это значит «стоп»?

— Нет, не «стоп», а переход на строку 5. Тут проверка: равен ли нулю признак R? Не равным нулю он может оказаться, если мы записывали новое распоряжение — видите в строке 12 присваивание R = 1, да? — или снимаем с контроля одно из старых — вы, наверное, уже заметили такое же присваивание в строке 13. Тогда надо переписать всю картотеку заново. Игнорируем в логическом операторе IF R = 0 THEN 6 переход на строку 6 и продолжаем выполнять строку 5 — записываем на диск всю картотеку, все массивы A ⊗ (), D ⊗ (), D(), S ⊗ (), T() и дату последней правки D0 ⊗. Если же R = 0, то переходим на указанную при этой проверке строку 6. Вот тут останавливаемся: STOP.

— Хорошая программа, — подытожил ее разбор НП. — Запускай! — потребовал он.

— Запускать еще рано, — возразил я. — Возникнет программная ошибка с аварийным остановом на второй строке.

— Ошибка? Какая?

— Попытка считывания с диска несуществующего файла «КАРТОЧКИ». Этот файл нужно еще организовать.

Я набрал на клавиатуре: DATA SAVE DC OPEN R (100) «КАРТОЧКИ».

НП сел за клавиатуру и стал заполнять одну карточку за другой. Мне оставалось лишь удивляться его памяти: он отлично помнил, когда, кому и какое распоряжение отдал.

После того как мы завели электронную картотеку распоряжений начальства и своих собственных насущных дел, наши традиционные утренние пятиминутки стали проходить совсем по-иному: они на самом деле стали пятиминутками. Мы включаем машину, загружаем в нее программу и картотеку, а затем работаем с ней. Сначала мы просматриваем, что нужно выполнить сегодня и необходимо сделать в ближайшие дни. Исходя из этого и с учетом новых задач мы составляем или корректируем планы на будущее — записываем что-то за конкретным исполнителем, снимаем с контроля выполненное. Для наиболее забывчивых и неорганизованных распечатываем на принтере то, что за ними числится.

Наш НП преобразился, стал отдавать больше времени науке. На вопросы о внутренней жизни лаборатории он теперь все чаще отвечает: «Спросите у машины».

Блочная структура программ

Программа должна состоять из отдельных, четко разграниченных блоков, те в свою очередь — из блоков более мелких и так до отдельных операторов.

Это требование структурного программирования, как мы видим на примере электронной картотеки приказов (см. рис. 4.6), оборачивается еще одним достоинством, носящим чисто психологический характер: такие программы легко объяснять заказчику, в связи с чем он бывает доволен исполнением заказа.

Печать на принтере

Что делать, если увиденную на дисплее информацию хочется зафиксировать на бумаге с помощью принтера (рис. 4.7). На «Искре 226» этой цели служит оператор SELECT PRINT/0C. Чтобы дальнейшая информация вновь выводилась оператором PRINT на дисплей, а не на бумагу, нужно выполнить оператор SELECT PRINT / 05. Для того чтобы сделать листинг программы, вывести значения переменных и констант, используются операторы LIST/0C и PRINT/0C (в других версиях — LLIST и LPRINT).

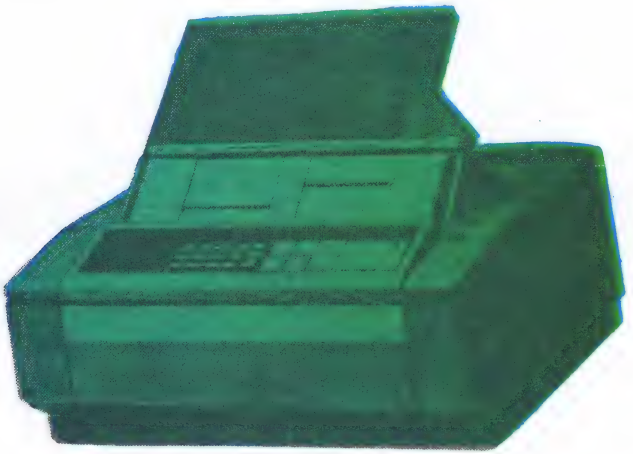


Рис. 4.7. Принтер

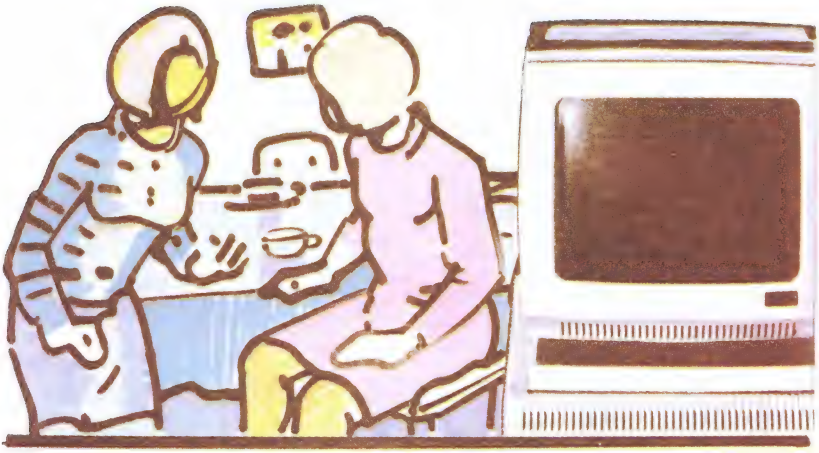
Версия Бейсика, с которой работают машины ДВК-2М и СМ-4, не позволяет вывести данные на печать столь же простым путем. Эти данные сначала нужно записать на магнитный диск в виде файла, а потом уже переправить файл на принтер.

Блоки программы и блоки компьютера

Одно поколение компьютеров приходит на смену другому, меняются господствующие стили программирования — и каждый раз можно усмотреть глубокие аналоги между основными концепциями, которыми руководствуются и создатели компьютеров, и работающие на этих компьютерах программисты.

Лозунгом дня стало сегодня структурное программирование. Хорошо структурированная программа представляется системой вложенных друг в друга модулей, у каждого из которых один вход и один выход. Тот, кто овладел навыками составления таких программ, быстро пишет их, нередко используя уже готовые стандартные модули, быстро редактирует и отлаживает. В этом он подобен конструктору современной электронной аппаратуры, состоящей из отдельных самостоятельных блоков — больших интегральных схем, блоков питания и т. д. Их число не так уж велико, но из них можно собирать разнообразные приборы, а в уже собранных — легко искать неисправности.

Неструктурированные программы подобны старым электронным схемам, где простейшие элементы (резисторы, конденсаторы, диоды, транзисторы и прочее) связаны между собой тысячами проводов. Собирать такую схему, искать в ней неисправности чрезвычайно трудно.



Этюд 22. К ВАМ НЕОЖИДАННО НАГРЯНУЛИ ГОСТИ...

— Компьютер в доме? Сущее наваждение! С тех пор, как он у нас появился, я мужа своего целыми днями не видела. Приходит с работы — и в свою комнату, к компьютеру своему ненаглядному. Сидит с ним до вечера, а то и заполночь, а то и до третьих петухов. Хорошо, что в последнее время немного одумался. Нет, от компьютера не отходит, что ты! Но хоть программы пишет не какие-то бестолковые, а в хозяйстве полезные. Вот «Поваренная книга», например. Хочешь покажу? Да что ты! Конечно, не сломаю! Я ведь потихоньку от своего благоверного тоже машину освоила. Ставим диск, включаем, загружаем программу...Смотри, вот она на дисплее (рис. 4.8). Все понятно, правда? Ты ведь у себя на работе курс лекций по Бейсику прослушала, не то, что я — по книжкам, урывками.

Строка 1 — описание массивов и литерных переменных, строка 2 — считывание файла «КУХНЯ». Кстати, тоже мне файл — в нем всего-то одно число F. Это сколько рецептов в книге. Так в строке 3 и выводится на дисплей: в поваренной книге столько-то рецептов.

Строка 4 — меню. Выбираю режим работы. Допустим, A = 1. Запись нового рецепта. Оператор IF A = 2 THEN 15 не срабатывает, идем по порядку на строку 6. С этой строки и по 12-ю происходит запись: название блюда, число составляющих, их названия, количество, сам рецепт, число порций, потом число рецептов в книге. Записали — возвращаемся на строку 3. На дисплее пишется, сколько теперь рецептов в книге. Потом переходим на строку 4, опять работаем с меню.

```
1  DIM I¤(30),N¤(30),N(30),R¤(30),N¤50,B¤50,R¤250
2  DATA LOAD DC OPEN R"КУХНЯ":DATA LOAD DC F
3  PRINT "В ПОВАРЕННОЙ КНИГЕ";F;"РЕЦЕПТОВ"
4  INPUT "1- ЗАПИСЬ НОВОГО РЕЦЕПТА,2-ПОИСК РЕЦЕПТА ПО
   НАЗВАНИЮ,3-ПО ИМЕЮЩИМСЯ ПРОДУКТАМ,4-КОНЕЦ",A:
   ON A GOTO 5,5,13:STOP
5  INPUT "НАЗВАНИЕ БЛЮДА",N¤:IF A=2 THEN 15
6  INPUT "ЧИСЛО НЕОБХОДИМЫХ СОСТАВЛЯЮЩИХ",N
7  PRINT "ВВЕДИТЕ ИХ НАЗВАНИЕ,НУЖНОЕ КОЛИЧЕСТВО,
   ЕДИНИЦЫ ИЗМЕРЕНИЯ (КГ,ШТ,МЛ,ЛОЖКИ И Т.Д.)"
8  FOR J=1TON:PRINT J;".";:INPUT N¤(J),N(J),R¤(J):NEXT J
9  R¤=" ":LINPUT "РЕЦЕПТ ПРИГОТОВЛЕНИЯ",R¤:INPUT "НА
   СКОЛЬКО ПОРЦИЙ ",P
10 F=F+1:CONVERT F TO F¤,(########)
11 DATA SAVE DC OPEN R(10)F¤:DATA SAVE DC N¤,N,N¤(),
   N(),R¤(),R¤,P:DATA SAVE DC END
12 SCRATCH R"КУХНЯ":DATA SAVE DC OPEN R("КУХНЯ")"КУХНЯ"
   :DATA SAVE DC F:DATA SAVE DC END:GOTO 3
13 INPUT "ЧИСЛО ИМЕЮЩИХСЯ ПРОДУКТОВ",N1
14 PRINT "ВВЕДИТЕ ИХ НАЗВАНИЕ":FOR J=1 TO N1:PRINT J;
   ".";:INPUT I¤(J):NEXT J
15 B=0:FOR I=1 TO F:CONVERT I TO F¤,(########)
16 DATA LOAD DC OPEN R F¤:DATA LOAD DC B¤,N,N¤(),N(),
   R¤(),R¤,P
17 IF A=3 THEN 18:IF N¤=B¤ THEN 19:GOTO 20
18 FOR J=1 TO N1:FOR K=1 TO N:IF I¤(J)=N¤(K) THEN 19:
   NEXT K:NEXT J:GOTO 20
19 B=1:PRINT B¤;:FOR J=1 TO N:PRINT J;".";N¤(J);N(J);
   R¤(J):NEXT J:PRINT "РЕЦЕПТ-";R¤;" НА ";P;"ПОРЦИЙ"
20 NEXT I:REM СЛЕДУЮЩИЙ РЕЦЕПТ
21 IF B=1 THEN 4:PRINT "ТАКОГО В КНИГЕ НЕТ":GOTO 4
```

Рис. 4.8. Программа «Поваренная книга»

Пусть теперь A = 2. Поиск рецепта по названию. Начинается, как раньше, со строки 5. Ввожу название блюда. И по условию IF A = 2 иду на строку 15. Тут сразу присваивание B = 0. Это на случай, если рецепта с таким названием в книге не найдется. Ищется он в цикле, номера I перебираются от 1 до F, и каждый раз считывается файл с таким номером. На строке 17 — опять проверка. Если A = 2, а я такое A и задала,и N ⊗ = B ⊗, то перехожу на строку 19. Она начинается с присваивания B = 1. Потом на дисплей выводится рецепт. На строке 20 число I увеличивается на единицу: если в книге есть еще какой-то рецепт с таким названием, надо искать его дальше. Когда I равно F, поиск заканчивается, все рецепты просмотрены. На строке 21 — еще одна проверка. Если B = 1, значит хоть один нужный рецепт в книге нашелся. Возвращаемся на строку 4, к меню. Если B = 0, тоже возвращаемся туда же. Но на дисплее тогда будет надпись: нужного рецепта в книге нет.

Ты, конечно, скажешь, что разумнее искать рецепты не по названиям, а по имеющимся продуктам, не по тому, что есть в книге, а по тому, что есть в доме — кусок баранины, филе трески,

кочан капусты. Такой режим в меню тоже есть: A = 3. Тогда — переход на строку 13. Здесь ввод числа продуктов. Дальше на строке 14 — ввод их названий в цикле. На строках 15 и 16 — считывание рецептов с диска, тоже в цикле. Считала какой-то I-й рецепт, иду на строку 17. Сейчас A = 3, переходим со строки 17 на следующую, 18-ю. Тоже непростая строка. Два цикла, вложенных друг в друга. По J перебираются все продукты, которые я назвала, по K — все продукты, которые есть в I-м рецепте. Совпадут хотя бы раз названия этих продуктов — перехожу на строку 19: IF I ⊗ (J) = N ⊗ (K) THEN 19. На дисплей выводится рецепт, я его читаю. Пока читаю, машина переходит на строку 20. Сюда же она переходит, если не было ни одного совпадения I ⊗ (J) = N ⊗ (K). Со строки 20 выполняется переход на строку 15, но уже с новым значением I. Считываю с диска рецепт с этим новым номером. И опять: если есть в нем хотя бы один продукт, который я задала, такой рецепт выводится на дисплей. При каждом выводе выполняется присваивание B = 1. Это опять для того, чтобы при B = 1 вернуться на строку 4, к меню, с одним или несколькими рецептами на дисплее, без примечания, а при B = 0 — с примечанием: нет в книге таких рецептов.

Вот и все. Что тут трудного? Или ты думаешь, что работа с персональным компьютером — привилегия мужчин?

Что недосмотрела хозяйка!

Прочтите еще раз рассказ хозяйки. Не замечаете ли вы в нем каких-либо ошибок, неточностей? Нет? Тогда вчитайтесь внимательнее в ее комментарий к строке 16 программы. Верно ли выражение «файл с таким номером»? Не совсем. Имя файла — это всегда литерная переменная. Не случайно номер I переводится в строке 15 в литерную форму с помощью оператора CONVERT. То же самое делается в строке 10 для записи нового рецепта. Однако хозяйка не сочла нужным останавливаться на превращениях числовой переменной F в литерную F ⊗.

Файлы-программы и файлы данных

Способы записи программ на магнитный диск и их считывания с диска отличаются от приемов, с помощью которых компьютер обменивается с диском информацией, необходимой для выполнения программ.

Вводя в память ЭВМ текст новой программы, следует предварительно выполнить команду NEW (новая; ею оперативная память тотчас очищается) и далее набрать имя, присваиваемое вводимой программе. Чтобы перенести ее на диск, достаточно затем отдать команду SAVE. Если после этого набрать новое имя, то именно под ним программа будет записана на диск отдельным файлом. Если новое имя не набрано, то программа сохранит то, которое указывалось прежде, вслед за выполнением команды NEW.

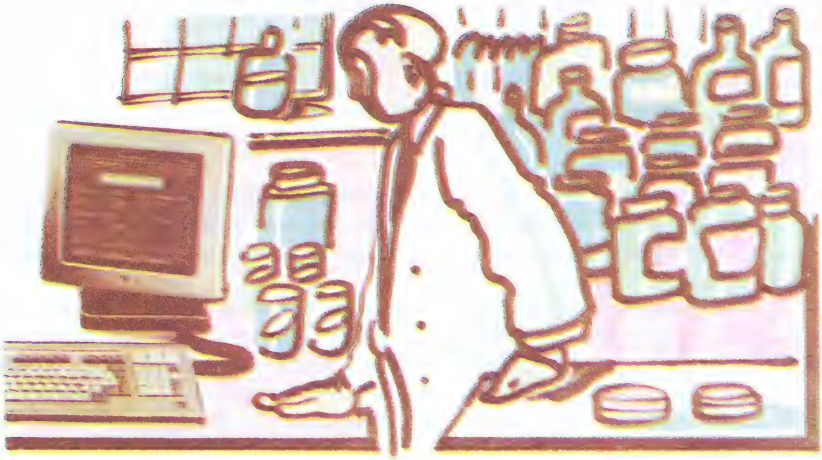
Существующая на внешнем устройстве программа вызывается в оперативную память командой OLD (старая), вслед за которой указывается имя программы. Перед ее считыванием оперативная память очищается.

Из-за различия в записи и считывании файлов, содержащих тексты программ и необходимые для их работы данные, говорят о файлах-программах и файлах данных.

Компьютер в семье бытовой техники

Домашний компьютер, войдя в семью бытовой техники, устанавливает тесные деловые отношения со своими «родственниками». Телевизор служит для него дисплеем, магнитофон — накопителем информации, печатная машинка — принтером.

Не остается без работы и телефон. С помощью специального устройства, называемого модемом, компьютер способен воспринимать и передавать по телефону информацию другим компьютерам. Так создается сеть ЭВМ, сильно повышающая возможности связанных ею компьютеров. Подключившись к ней, можно просматривать программы в чьем-то каталоге (конечно, с разрешения хозяина), переписывать их, использовать различные базы данных, справочно-информационные системы и т. д.



Этюд 23. КАК Я ЗАВЕДОВАЛ СКЛАДОМ ХИМРЕАКТИВОВ

История эта началась, когда мне, младшему научному сотруднику одного НИИ химического профиля, поручили совмещать свои прямые обязанности с заведованием складом реактивов нашей научной лаборатории.

Мои дополнительные обязанности состояли в том, чтобы я оперативно удовлетворял заявки коллег и своевременно пополнял запасы реактивов через отдел снабжения института.

Завел я себе общую тетрадь и решил вести в ней записи по реактивам — сколько чего осталось, где они хранятся и прочее. Мне думалось, что это решит все проблемы — посмотрел в алфавитном указателе номер страницы, где записаны данные по затребованному реактиву, и нашел его. Но оказалось все намного сложней. По мере того как записей становилось все больше и больше, работать с учетной тетрадью становилось все труднее. Произошел какой-то информационный бум. Во-первых, у реактива может быть много названий — от химической формулы до термина, который был в ходу еще у алхимиков. Обычная поваренная соль — это и хлорид натрия, и натрий хлористый, и каменная соль, и галит, и, наконец, NaCl. Во-вторых, часто нужно найти не какой-то конкретный реактив, а что-то удовлетворяющее чему-то. Например, какую-нибудь соль серной кислоты для приготовления раствора с сульфат-ионами. Подходящих реактивов для этого может быть много. Какой лучше взять?

В конце концов стало ясно: какую систему записей ни заводи — по алфавиту ли, по содержимому шкафов и полок, по степени чистоты реактивов, — так или иначе при поиске

нужной упаковки приходилось просматривать всю тетрадь от корки до корки, выискивая на ее страницах требуемое сочетание символов: «Cl», «ХЛОР», «НАТР» и т. д. (Такие цепочки букв и других символов называют ключевыми словами поиска.)

Идея учетной тетради зашла в тупик. Проще было сразу поискать реактив по шкафам. Информационный бум перерос в информационный кризис.

Тут наша лаборатория получила персональную ЭВМ с языком программирования Бейсик. Его я освоил довольно быстро и понял, что такой компьютер — идеальное средство для ведения учета химреактивов.

Я составил программу, которая превращает машину в электронную учетную книгу. В качестве долговременного носителя информации я решил использовать магнитофон — дисковода у нашей ЭВМ нет. За три дня перенес записи из своей старой потрепанной тетради на магнитную ленту. Заодно провел ревизию всех шкафов.

Есть в моей программе (рис. 4.9) два более или менее сложных места: обмен информацией между оперативной памятью ЭВМ и постоянным запоминающим устройством, а также поиск информации по ключевым словам.

Если данные по реактивам хранить только в оперативном запоминающем устройстве ЭВМ, то машину нельзя будет ни выключить на ночь, ни работать на ней по другой программе, так как вся ранее введенная информация сотрется. Никто не позволит, чтобы дорогая и дефицитная ЭВМ была только амбарной книгой. Эту проблему решает встроенный магнитофон. На обычной компакт-кассете можно хранить как программы, так и данные.

Теперь о работе по программе «Склад реактивов».

Строка 1. В учетной книге 200 страниц. В памяти машины резервируется место под хранение трех одномерных массивов — двух литерных и одного числового. Элементы литерного массива A ⊗ ( ) хранят записи в книге. Максимальная длина записи на одной странице — 255 знаков, включая пробелы. В массив K ⊗ ( ) записываются два ключевых слова для поиска нужной информации в учетной книге — одного слова часто бывает недостаточно. Массив B( ) хранит информацию о том, найдены или нет ключевые слова на странице. Если, например, B(1) = 1, а B(2) = 0, то это значит, что первое слово найдено, а второе нет.

Переменная R также может быть равна либо нулю, либо единице. Содержание учетной книги хранится и на магнитной ленте, и в памяти машины. Пока они тождественны, R = 0. Если в машине что-то изменилось, то R = 1, и нужно менять содержание книги на магнитной ленте.

Строка 2. ЭВМ переписывает содержание книги с магнитной ленты в свою оперативную память. Для этого книгу (файл с именем КНИГА) нужно открыть (OPEN) для считывания (R —

```
1 N=200:DIM A$(N),K$(2),B(2):R=0:REM ИНИЦИАЛИЗАЦИЯ
2 PRINT "ПОСТАВЬ КАССЕТУ И НАЖМИ СЧИТЫВАНИЕ":
  ROPEN"КНИГА":FOR I=1 TO N:INPUT/T A$(I):NEXT:CLOSE
3 INPUT "1-НОВАЯ ЗАПИСЬ, 2-ПОИСК, 3-ПЕРЕЛИСТЫВАНИЕ,
  4-ИЗМЕНЕНИЕ В ЗАПИСЯХ, 5-СТИРАНИЕ ЗАПИСИ, 6-КОНЕЦ";A:
  ON A GOTO 6,8,9,18,18,4
4 IF R=1 THEN PRINT "ПЕРЕМОТАЙ ЛЕНТУ И НАЖМИ ЗАПИСЬ.":
  WOPEN"КНИГА":FOR I=1 TO N:PRINT/T A$(I):NEXT:CLOSE
5 STOP :REM ОСТАНОВ ВЫПОЛНЕНИЯ ПРОГРАММЫ
6 FOR I=1 TO N:
  IF A$(I)="" THEN INPUT A$(I):R=1:GOTO 3
7 NEXT:
  PRINT "ЧИСТЫХ СТРАНИЦ НЕТ.":GOTO 3
8 INPUT "1-Е КЛЮЧЕВОЕ СЛОВО,СВЯЗКА (И,ИЛИ,НЕ),
  2-Е СЛОВО ?";K$(1),L$,K$(2)
9 C=0:
  FOR I=1 TO N:
  IF A$(I)="" THEN 16
10 IF A=3 THEN B=1:GOTO 15
11 FOR J=1 TO 2:
   B(J)=0:
   FOR P=1 TO LEN(A$(I))-LEN(K$(J))+1:
   IF K$(J)=MID$(A$(I),P,LEN(K$(J)))THEN B(J)=1:GOTO 13
12 NEXT P:REM НОВАЯ ПОЗИЦИЯ В ЗАПИСИ НА СТРАНИЦЕ
13 NEXT J:REM НОВОЕ КЛЮЧЕВОЕ СЛОВО:
   B=B(1)*B(2):REM СВЯЗКА И:
   IF L$="ИЛИ" THEN B=B(1)+B(2):GOTO 15
14 IF L$="НЕ" THEN B=B(1)-B(2)
15 IF B>0 THEN PRINT I","A$(I):C=1
16 NEXT I:REM НОВАЯ СТРАНИЦА:
   IF C=0 THEN PRINT "ИНФОРМАЦИИ НЕТ."
17 GOTO 3
18 INPUT "НОМЕР СТРАНИЦЫ ?";I:PRINT A$(I):A$(I)="":
   IF A=4 THEN INPUT A$(I)
19 R=1:GOTO 3
```

Рис. 4.9. Программа «Склад реактивов»

read), перелистать все ее страницы от первой до последней и ввести (INPUT/T) в память машины содержание каждой страницы A ⊗ (I). После этого книгу можно закрыть (CLOSE).

Строка 3. На экране дисплея появляется меню — перечень услуг, какие машина может нам оказать.

При составлении программы были предусмотрены пять возможных режимов работы с учетной книгой:

1 — новая запись на первую чистую страницу книги. Машина сама ее отыскивает. Если ее не окажется, то ЭВМ об этом даст знать. В этом случае нужно будет какую-нибудь страницу очистить (см. режимы 4 и 5);

2 — поиск нужных сведений;

3 — перелистывание книги;

4 — изменение записи в книге;

5 — стирание записи;

6 — конец. Такой пункт замыкает каждое меню.

После того как на дисплее появилось меню, ЭВМ прерывает выполнение программы и ждет, пока не будет нажата пара клавиш — с номером выбранного вида работы и ВК. Набранный номер присваивается переменной A, управление передается на нужный участок программы.

После выполнения всех заданий, кроме, конечно, шестого, машина снова выдаст на экран дисплея меню и будет готова работать с учетной книгой дальше.

Строка 4. Если содержимое учетной книги, хранящееся в оперативной памяти машины, изменилось (а это происходит после работы в режиме 1, 4 или 5), переменная R станет равна 1. Это будет сигналом для перезаписи содержимого книги на магнитную ленту. На дисплее появится просьба к пользователю перемотать ленту на начало записи и нажать на магнитофоне клавиши записи. При этом ЭВМ откроет (OPEN) для записи (W — *write*) файл с именем КНИГА, циклически запишет на диск все страницы книги (PRINT/T) от первой до двухсотой и закроет файл (CLOSE). После этого счет по программе прерывается — см. строку 5.

Строки 6 и 7. В них записаны операторы, позволяющие сделать новую запись на первой из чистых страниц книги.

Вот образцы таких записей.

1-я стр. Кислота серная аккумуляторная H2SO4 15 л норма 10 л вытяжной шкаф 2 13.08.88

2-я стр. Натр едкий технический сода каустическая NaOH 10 л норма 20 л меньше нормы вытяжной шкаф 2 02.04.88

3-я стр. Соль поваренная каменная галит натрий хлористый NaCl 4 кг норма 2 кг 5 шкаф 4 полка 09.06.88

4-я стр. Натрий хлористый ЧДА 0.2 кг меньше нормы 5 шкаф 3 полка норма 2 кг 20.10.88

5-я стр. Трилон Б комплексон III нет совсем меньше нормы норма 1 кг брали в долг у соседей 0.4 кг 10.09.88

6-я стр. Сульфат натрия ХЧ 3 кг норма 2 кг где-то лежит

Записи на страницах электронной амбарной книги должны быть похожи на тексты хорошо составленных телеграмм, которые понятны получателям и не очень дороги для отправителя.

Как уже говорилось, максимально возможная длина записи на одной странице книги — 255 знаков. Злоупотребление длинными записями замедляет поиск информации, увеличивает время записи и считывания.

На четвертой странице книги записано, например, что 20 октября 1988 года на складе химреактивов было всего 200 грамм хлористого натрия чистого для анализа. Это меньше нормального запаса — 2 кг. Соль хранится в пятом шкафу на третьей полке.

Строки 8 — 17 осуществляют поиск нужной информации в учетной книге. Как это делается, лучше пояснить заявками вида A, B и C:

Есть ли ключевое слово на странице?		Нужно ли выводить на дисплей содержание страницы?		
первое	второе	логическая связка		
		И	ИЛИ	НЕ
да	да	да	да	нет
да	нет	нет	да	да
нет	да	нет	да	нет
нет	нет	нет	нет	нет

Рис. 4.10. Таблица логических соотношений

А. Нужно подобрать какую-нибудь соль соляной кислоты, но так, чтобы реактив был химически чистый. Найти подходящую соль можно, если открыть книгу на страницах, где есть слова «␣ХЛОР» и «ХЧ». Пробел «␣» не даст книге открыться там, где есть слова «ЧЕТЫРЕХХЛОРИСТЫЙ» или «ДИХЛОР» и т. д. Поэтому я умышленно не разделял слова в записях на страницах книги знаками препинания, а ставил между ними только пробелы. С запятой нужно быть особенно осторожным, так как она отсекает ввод информации. Чтобы оператором INPUT ввести литерную переменную, содержащую запятые, нужно ее заключить в кавычки.

В. Реактив, как уже указывалось, может называться по-разному. Тот же хлористый натрий можно найти, открыв книгу на страницах со словом «ПОВАРЕННАЯ СОЛЬ» или «ГАЛИТ».

С. Если какого-то реактива осталось мало, то его нужно постараться не использовать, а подыскать замену, открыв книгу на страницах с фрагментом его названия, но там, где нет слова «МЕНЬШЕ НОРМЫ».

Эти три примера подсказали форму подачи ключевых слов поиска: первое слово, логическая связка (И, ИЛИ, НЕ), второе слово, ЭВМ выводит на экран дисплея содержание нужных страниц, руководствуясь таблицей, представленной на рис. 4.10.

Теперь при поиске реактивов по заявке вида А достаточно набрать на клавиатуре «ХЛОР, И, ХЧ», по заявке вида В — «ПОВАРЕННАЯ СОЛЬ, ИЛИ, ГАЛИТ», а по заявке вида С — «ХЛОР, НЕ,МЕНЬШЕ НОРМЫ». Для такого запроса программа останавливается на строке 8.

Если возникнет необходимость просмотреть книгу только по одному слову, то его нужно набрать дважды, отделив записи логической связкой И или ИЛИ.

Так, по ключевому слову «МЕНЬШЕ НОРМЫ» можно быстро составить заявку для отдела снабжения. Ее лучше отпечатать на принтере и отнести бумагу снабженцам. Отказов в этом случае,

как я убедился, будет меньше — заявку-то составляла ЭВМ, существо точное и беспристрастное.

По ключевому слову «5 ШКАФ» можно узнать, что в нем лежит, а ключевое слово «ГДЕ-ТО ЛЕЖИТ» может характеризовать работу заведующего складом химреактивов.

Фрагмент программы, управляющий поиском нужной информации, краток, но позволяет совершать довольно много операций.

На строках 9 — 16 организовано три цикла. Они вложены друг в друга, как матрешки. Первый внешний цикл с заголовком на строке 9 перелистывает страницы амбарной книги. Второй цикл с заголовком на строке 11 перебирает два ключевых слова, а в третьем цикле ведется проверка совпадения одного из них с фрагментом текста на странице книги. Переменная этого цикла P — номер знака в тексте.

Ключевое слово как бы движется вдоль текста на странице книги от первой до последней его буквы:

```
A ⊗ (I) = «АММИАК ВОДНЫЙ НЕТ СОВСЕМ 31.12.88»
K ⊗ (J) = «  ХЛОР»  ——→      «  ХЛОР»
```

У каждой буквы делается остановка с проверкой на совпадение. Например:

```
A ⊗ (I) = «НАТРИЙ ХЛОРИСТЫЙ»
K ⊗ (J) = «          ХЛОР»
```

Если совпадение обнаружено, то переменная B(J) становится равной единице (см. конец строки 11), нет — остается равной нулю.

Строки 13 — 15. После того как оба ключевых слова будут проверены, команды этих строк, руководствуясь таблицей логических соотношений, решат, нужно ли данную страницу книги выводить на дисплей.

Строка 16. Переменная C до начала поиска принимает нулевое значение (см. начало строки 9). В конце поиска она останется равной нулю лишь в том случае, если информация в книге не найдена, о чем машина сообщит пользователю (см. конец строки 16).

Строки 18 и 19 содержат операторы, реализующие 4-й и 5-й режимы работы, — изменение информации в книге.

Самый первый раз запустить программу нужно не командой RUN, а такими командами:

```
N = 200 : DIM A ⊗ (N), K ⊗ (2), B(2) : R = 1 : GOTO 3
```

Эту цепочку символов необходимо набрать на клавиатуре компьютера и затем нажать клавишу ВК.

Отличие в запусках программы в первый и последующий разы связано с тем, что вначале на магнитной ленте еще не было записи с данными по складу — нет файла под именем КНИГА.

Команды строки 4 эту запись создадут, и в следующий раз программу можно будет запустить обычным способом — командой RUN.

Как составить электронный справочник!

Кому не приходилось мучиться в поисках номера телефона в пухлой записной книжке, названия научной статьи — в картотеке, книги — в книжном шкафу? Только что разобранная программа способна навести порядок и тут. Она может работать и с телефонной книгой, и с картотекой научных статей, и с каталогом домашней библиотеки.

Для этого придется изменить лишь первую строку программы, где указывается число страниц учетной книги. Иными будут, разумеется, и записи в ней:

7756756 Виталий ремонт ЭВМ

Программа обработки данных метод наименьших квадратов кассета 5 позиция на счетчике 165

В. Скотт Айвенго исторический роман 5 полка дал почитать

Две группы операторов

В программе «Склад реактивов» (см. рис. 4.9) строка 11, пожалуй, самая интересная и сложная. В ней определяется, является ли ключевое слово частью строки, записанной на очередной странице тетради. Такая процедура является ядром многих справочно-информационных систем и программ. Немудрено, что во многих версиях Бейсика есть специальная функция, упрощающая данную процедуру:

B = INSTR (P, A ⊗, K ⊗)

После выполнения этого оператора числовая переменная B примет значение, зависящее от того, обнаружена или не обнаружена литерная переменная K в литерной переменной A, если та просматривается, начиная с P-й позиции. Если обнаружена, B выразит собой номер первого символа строки K ⊗ в строке A ⊗, если не обнаружена, B будет равна нулю.

Такого оператора нет в той версии Бейсика, на которой написана программа «Склад реактивов», и его пришлось заменить комбинацией операторов, имеющихся в использованной версии.

Все операторы Бейсика условно можно поделить на две группы. В первую из них (*ядро языка*) входит необходимый минимум программных средств. Операторы этой группы следует хорошо заучить и освоить, поскольку они применяются в каждой версии Бейсика. Вторая группа операторов содержит дополнительные средства, облегчающие программирование. В тех версиях, где какой-то из этих операторов отсутствует, его приходится заменять комбинацией имеющихся. Операторы второй группы специально заучивать не стоит — они сами запоминаются, если часто оказываются кстати.

Этюд 24. ПОСЛЕДНЕЕ ЗАДАНИЕ

Свою книгу о программировании на Бейсике я решил закончить небольшим экзаменом. Сначала думал предложить читателю несколько вопросов, каждый с несколькими вариантами ответов, из которых читатель выбрал бы правильный. Например:

А. Укажите, какое число появится на дисплее после выполнения операторов:

```
FOR I = 1 TO 6 STEP 1.5 : NEXT I : PRINT I
```

1. Семь
2. Пять с половиной
3. Шесть
4. Не знаю. Это зависит от того, какая версия Бейсика применяется. Опытный программист никогда не станет использовать параметр цикла сразу после выхода из него.

Б. Чему будет равна максимальная длина литерной переменной при работе на ЭВМ «Искра 226», если это не оговорено оператором DIM?

1. 16 байт
2. 8 байт
3. 64 байта

В. Какое число появится на дисплее, если выполнить операторы

```
A = 4.55 : PRINT USING «#. #», A
```

1. 4.5
2. 4.6
3. Не знаю. Это зависит от особенностей используемой версии Бейсика. В одних версиях последняя из цифр, выводимых на дисплей оператором PRINT USING, округляется, в других все цифры, выходящие за рамки указанного формата, отбрасываются.

Список подобных вопросов можно продолжить, а потом предложить изучающему Бейсик ответить на них. По числу правильных ответов можно судить о качестве знаний языка. В этом и заключается так называемый программируемый экзамен.

Надо сказать, что некоторые опытные методисты отвергают подобную манеру проверки знаний. Они утверждают, что ученикам нельзя показывать заведомо неверные ответы, поскольку иногда они запоминаются лучше верных. Несмотря на эти опасения, программируемые экзамены начали широко применяться в школах и вузах еще задолго до «эры компьютеризации». Вычислительная техника дала этому процессу новый импульс. Были при этом учтены и возражения опытных методистов. Рекомендуется вслед за каждым из задаваемых вопросов предлагать такие ответы, чтобы среди них один был совершенно правильным, а другие содержали весьма незначительные неточности. Такие ответы и прилагались к вышеприведенным вопросам A и B.

Свой вклад в компьютеризацию экзаменов захотелось сделать и мне. Я предлагаю программу на Бейсике для ЭВМ «Искра 226», позволяющую легко и быстро создать «автоматизированную систему обучения чему угодно» (рис. 4.11).

Все вопросы хранятся на магнитном диске. Их общее число также записано на диске отдельным файлом с именем ВОПРОСЫ. Оно считывается операторами строки 2 и выводится на дисплей (см. строку 3).

Меню (строка 4) подскажет, что можно потребовать от машины: 1 — записать на диск новый вопрос; 2 — подправить неудачный старый (новый вопрос запишется на диск новым файлом, а исправленный вариант старого — на прежнее место); 3 — провести экзамен. ЭВМ будет считывать с диска случайно попавшийся вопрос и задавать его экзаменуемому (строки 18 — 19), требуя нажать клавишу с номером правильного ответа. Машина будет также реагировать на нажатие клавиш с номерами неверных ответов, а остальные клавиши будут заблокированы (строка 22). «Пересидеть» машину нельзя, так как время на обдумывание ограничено и превышение его равносильно неверному ответу (строка 21).

Впрочем, к чему комментировать программу? Разобраться в ней вполне по силам тому, кто вдумчиво прочтет мою книгу. Такой разбор пусть и станет для читателя первым экзаменационным заданием. Задание второе — придумать вопросы и варианты ответов, записать их на магнитный диск и проэкзаменовать с помощью ЭВМ своих знакомых, также изучавших Бейсик, или своих учеников, если таковые имеются.

Есть опасение, что иной ученик может прервать работу программы-экзаменатора, подсмотреть правильный ответ, распеча-

```
1  DIM V¤200,O¤(5)100,M¤,B(50)
2  M¤="ВОПРОСЫ":GOSUB 26:DATA LOAD DC M
3  PRINT "ВОПРОСОВ НА ДИСКЕ -";M
4  INPUT"1-НОВЫЙ ВОПРОС,2-ИСПРАВЛЕНИЕ,3-ЭКЗАМЕН,4-КОНЕЦ"
   ,A:ON A GOTO 5,15,16:STOP
5  M=M+1:K=M:GOSUB 27:PRINT "ВОПРОС";M:V¤=" "
6  LINPUT V¤:IF A=2 THEN 9
7  INPUT "ЧИСЛО ВАРИАНТОВ ОТВЕТА (2-5)",N:IF N<2 OR N>5
   THEN 7:FOR I=1 TO N:O¤(I)=" ":NEXT I
8  INPUT "НОМЕР ПРАВИЛЬНОГО ОТВЕТА",P:IF P>N THEN 8
9  FOR I=1 TO N:PRINT I;"-Й ВАРИАНТ ОТВЕТА (";:IF I=P
   THEN 10:PRINT "НЕ";
10 PRINT "ВЕРНЫЙ)":LINPUT O¤(I):NEXT I
11 IF A=2 THEN 12:DATA SAVE DC OPEN R(7) M¤:GOTO 13
12 GOSUB 28
13 DATA SAVE DC V¤,N,P,O¤():GOSUB 29:IF A=2 THEN 3
14 M¤="ВОПРОСЫ":GOSUB 28:DATA SAVE DC M:GOSUB 29:GOTO 3
15 INPUT "НОМЕР ВОПРОСА",K:IF K>M THEN 15:GOSUB 27:
   GOSUB 26:GOSUB 30:GOTO 6
16 INPUT"ЧИСЛО ВОПРОСОВ",V:IF V>M OR V>50 THEN 16:
   MAT B=ZER:R=0:INPUT"ВРЕМЯ НА ОТВЕТ (СЕК)",S
17 FOR J=1 TO V
18 K=1+INT(V*RND(1)):IF B(K)=1 THEN 18:B(K)=1:GOSUB 27:
   GOSUB 26:GOSUB 30:CONVERT P TO P¤,(#):CONVERT N
   TO N¤,(#)
19 PRINT HEX(03);"ВОПРОС -";J:PRINT STR(V¤,1,LEN(V¤)):
   PRINT:FOR I=1TON:PRINT I;".";STR(O¤(I),1,LEN(O¤(I)))
   :NEXT I:INPUT ¤TO
20 A¤=" ":KEYIN A¤,22,20
21 INPUT ¤T:IF (TO-T)/2000<S THEN 20:PRINT "ВРЕМЯ НА
   ОБДУМЫВАНИЕ ИСТЕКЛО.":GOTO 25
22 IF A¤<"1" OR A¤>N¤ THEN 20
23 IF A¤=P¤ THEN 24:PRINT "НЕВЕРНО.":PRINT "ПРАВИЛЬНЫЙ
   ОТВЕТ -";O¤(P):GOTO 25
24 PRINT "ПРАВИЛЬНО.":R=R+1
25 FOR L=1 TO 1000:NEXT L:NEXT J:PRINT " ВЫ ОТВЕТИЛИ
   ПРАВИЛЬНО В ";R;" СЛУЧАЯХ ИЗ ";V:GOTO 3
26 DATA LOAD DC OPEN R M¤:RETURN
27 CONVERT K TO M¤,(########):RETURN
28 SCRATCH R M¤:DATA SAVE DC OPEN R (M¤)M¤:RETURN
29 DATA SAVE DC END:RETURN
30 DATA LOAD DC V¤,N,P,O¤():RETURN
```

Рис. 4.11. Программа «Экзаменатор»

тав массив O ⊗ ( ), и вновь запустить программу с того же места, на котором ее выполнение было приостановлено. Но если это случится на экзамене по программированию на Бейсике, то такого ученика незачем и контролировать — он уже достаточно хорошо знает предмет, по которому сдает экзамен.

ОПЕРАТОРЫ БЕЙСИКА В РАЗЛИЧНЫХ ВЕРСИЯХ

1. Присваивание значения переменной:
   A = B LET A = B A,B = C
2. Разделение операторов на строке:
   A = B : C = D\E = L|D = B
3. Условные переходы:
   IF A<>0 THEN GOTO 10
   IF A><0 GOTO 10
   IF A<>0 THEN 10
   IF A#0 GOTO 10
   IF NOT A GOTO 10
4. Логическое умножение:
   IF (A = B) * (C = 0) THEN . . .
   IF A = B AND C = 0 THEN . . .
   IF A = B THEN IF NOT (C) THEN . . .
5. Логическое сложение:
   IF (A> = B) + (C = D) GOTO 10
   IF A = >B OR C = D THEN 10
   IF A> = B THEN 10 : IF C = D THEN 10
6. Вывод переменных на дисплей:
   PRINT A ?A WRITE A DISP A SELECT PRINT 05 : PRINT A
7. Вывод переменных на принтер:
   LPRINT A PRINT/0C, A SELECT PRINT 0C : PRINT A
8. Ввод значения переменной с клавиатуры:
   INPUT «A?»; A INPUT«A»,A INPUT «A»A
   PRINT «A»;\INPUT A
9. Прием байта с клавиатуры без прерывания:
   GET A GET A ⊗ KEYIN A ⊗ 10,20 A ⊗ = GET ⊗ A ⊗ =KEY ⊗
10. Выставление курсора на дисплее:
    LOCATE X,Y CURSOR X,Y CUR X,Y
    VCURX : HCURY PRINT AT (X,Y)
11. Печать с форматом:
    PRINT USING«##. ##»,A PRINT!2.2!A
    A = INT (100 * A)/100 : PRINT A
12. Выделение части литерной переменной:
    MID ⊗ (A ⊗,A,B) STR(A ⊗,A,B) SEG ⊗ (A ⊗,A,A + B)
    LEFT ⊗ (A ⊗,A) RIGHT ⊗ (A ⊗,B)
13. Считывание программы с внешнего носителя:
    OLD APPEND LOAD MERGE
14. Опрос таймера:
    T = TIME T ⊗ = TI ⊗ T = CLK INPUT ⊗ T

15. Перевод литерной переменной в числовую:
    CONVERT A ⊗ TO A A = VAL (A ⊗)
16. Перевод числовой переменной в литерную:
    CONVERT A TO A ⊗,(#) A ⊗ = STR ⊗ (A)
17. Альтернатива:
    10 IF A = 0 THEN B = C: GOTO 30
    20 B = D
    30 . . .
    10 IF NOT (A) THEN B = C ELSE B = D
    10 IF NOT (A) THEN . . .: . . .: . . . ELSE . . .: . . . IFEND
18. Цикл «пока» (итерация с предпроверкой):
    10 IF A = 0 THEN . . .: GOTO 10
    10 WHILE NOT (A): . . .: WEND
19. Цикл «до» (итерация с постпроверкой):
    10 . . .: IF A = 0 GOTO 10
    10 REPEAT : . . .:UNTIL A〈〉 0
20. Поиск подстроки в строке:
    INSTR() POS ()
21. Стирание экрана дисплея:
    PRINT HEX (03) PRINT CHR ⊗ (22) CLS
22. Перевод курсора в верхний левый угол:
    PRINT HEX (00) PRINT CHR ⊗ (21) HOME
23. Перенумерация строк программы:
    RENUMBER RENUM RESEQ

ПЕРЕЧЕНЬ СЛУЖЕБНЫХ СЛОВ, ИСПОЛЬЗУЕМЫХ В ПРОГРАММАХ КНИГИ*

1. ABS — заголовок встроенной функции вычисления абсолютного значения переменной или алгебраического выражения, стоящего в скобках.
2. AND — операция логического умножения двух булевых переменных или выражений, стоящих справа и слева от слова AND; результат операции «верно», если верны оба булевых выражения, и «неверно», если одно или оба выражения неверны.
3. CHR — заголовок встроенной функции вызова символа, десятичный код которого записан в скобках.
4. CLOSE — оператор закрытия файла после записи в него данных (см. синоним ниже в п. 7е).
5. CONVERT — заголовок оператора перевода данных из числовой формы в литерную и обратно.
6. CURSOR — заголовок оператора перемещения курсора в заданную позицию экрана дисплея.
7. DATA — служебное слово, входящее в операторы:
   а) DATA + список констант, отделенных запятыми, — список данных в программе;
   б) DATA LOAD DC — считывание данных из файла;
   в) DATA LOAD DC ... OPEN (см. синоним в п. 49) — открытие файла для считывания;
   г) DATA SAVE DC — запись данных в файл;
   д) DATA SAVE DC ... OPEN (см. синоним в п. 64) — открытие файла для записи;
   е) DATA SAVE DC END — закрытие файла после записи.
8. DC — слово в операторах обмена данных между ОЗУ и дисководом, означающее, что диск (D) размечен для работы в режиме каталога (C) (см. операторы п. 7).
9. DEF FN — заголовок оператора, объявляющего в программе функцию пользователя.
10. DIM — заголовок оператора объявления массивов.
11. ELSE — служебное слово, стоящее в заголовке одного плеча альтернатив (заголовок второго плеча см. в п. 58).
12. END — а) в программах прерывает прогонку программы;
    б) см. оператор в п. 7е.
13. ERROR — служебное слово, входящее в операторы, предназначенные для включения и выключения анализатора программных ошибок.
14. F — символ, входящий в операторы обмена информацией между ОЗУ и дисководом, означающий, что задействован диск в кармане F (верхнем) дисковода (см. п. 7).

---

* Это не словарь операторов языка программирования Бейсик, а только расшифровка служебных слов. (Под Бейсиком понимается одна из его версий.)

15. FN — заголовок оператора вызова функции пользователя.
16. FOR — первое служебное слово в заголовке цикла с параметром.
17. GET — заголовок оператора приема байта с клавиатуры без прерывания (см. синоним в п. 22).
18. GOSUB — заголовок оператора перехода к подпрограмме.
19. GOTO — заголовок оператора безусловного перехода.
20. HEX — заголовок функции вызова символа, шестнадцатеричный код которого записан в скобках: 01 — перевод курсора в левый верхний угол экрана, 03 — то же, с очисткой экрана, 07 — подача звукового сигнала.
21. IF — заголовок оператора условного перехода или альтернативы.
22. INKEY — заголовок оператора приема байта с клавиатуры без прерывания (см. синоним в п. 17).
23. INPUT — заголовок оператора передачи информации в ОЗУ через клавиатуру.
24. INPUT ⊗ — заголовок оператора опроса встроенных часов ЭВМ.
25. INT — заголовок функции определения ближайшего большего целого числа.
26. LEFT — заголовок функции «вырезания» левой части литерной переменной.
27. LEN — заголовок функции определения длины литерной переменной.
28. LET — заголовок оператора присвоения значения переменной (в программах это слово обычно опускают).
29. LINE — служебное слово, входящее в оператор передачи через клавиатуру ЭВМ в ее ОЗУ цепочки символов, включая и знаки-разделители.
30. LINPUT — заголовок оператора коррекции значения литерной переменной.
31. LOAD — служебное слово, входящее в операторы передачи информации в ОЗУ машины из внешней памяти (см. п. 7).
32. LOG — заголовок оператора вычисления логарифма.
33. MAT — служебное слово, входящее в матричные операторы.
34. MID — заголовок функции «вырезания» середины литерной переменной (см. ниже синоним в п. 56).
35. NEXT — заголовок оператора фиксации конца цикла с параметром.
36. NOT — заголовок логической функции изменения значения булевой переменной на противоположный.
37. ON — а) заголовок оператора множественного ветвления;
б) служебное слово, входящее в оператор включения анализатора программных ошибок (см. п. 13).
38. OPEN — служебное слово, входящее в операторы открытия файлов (см. п. 7).
39. OR — операция логического сложения булевых переменных или выражений, стоящих справа и слева от слова OR; результат операции «верно», если верны оба или одно из двух булевых выражений.
40. POS — заголовок функции поиска подстроки в строке.
41. PRINT — заголовок оператора выдачи информации на экран дисплея, принтер или другое внешнее устройство (магнитная лента, например).
42. R — символ, входящий в операторы обмена информацией между ОЗУ и дисководом, означающий, что задействован диск в кармане R (нижнем) дисковода (см. синоним в п. 7).
43. READ — служебное слово, входящее в операторы присвоения переменным значений, считанных из блока данных.
44. REM — начало комментария.

45. RESTORE — оператор (заголовок оператора) возврата указателя счетчика считывания значений переменных из блока данных.
46. RETURN — оператор возврата из подпрограммы в основную программу или в подпрограмму предыдущего уровня.
47. RIGHT — заголовок функции «вырезания» правой части литерной переменной.
48. RND — заголовок функции, вызывающей псевдослучайное число.
49. ROPEN — заголовок оператора открытия файла для считывания (см. синоним в п. 7в).
50. SAVE — служебное слово, входящее в операторы передачи информации из ОЗУ машины во внешнюю память (см. синоним в п. 7).
51. SCRATCH — заголовок оператора присвоения файлу статуса ликвидированного.
52. SIZE — системная переменная, хранящая значения остатка памяти машины, свободной от программ и данных.
53. SQR — заголовок функции извлечения квадратного корня.
54. STEP — служебное слово в программах, указывающее шаг изменения параметра в цикле.
55. STOP — оператор, прерывающий выполнение программы.
56. STR — заголовок функции «вырезания» середины литерной переменной (см. синоним в п. 34).
57. SWAP — заголовок оператора, осуществляющего взаимный обмен значений у пары переменных.
58. THEN — служебное слово, стоящее в заголовке одного плеча альтернативы (заголовок второго плеча — см. синоним в п. 11).
59. TO — служебное слово в программах, указывающее предельное значение параметра цикла.
60. USING — служебное слово, указывающее формат печати при выводе значений числовых переменных на дисплей или принтер.
61. VAL — заголовок функции определения десятиричного кода символа, указанного в скобках.
62. WEND — служебное слово, замыкающее цикл «пока» в программах.
63. WHILE — заголовок цикла «пока» в программах.
64. WOPEN — заголовок оператора открытия файла для записи (см. синоним в п. 7д).

СЛОВАРЬ ТЕРМИНОВ

АВАРИЙНЫЙ ОСТАНОВ (АВОСТ) — прекращение выполнения программы, вызванное содержащимися в ней ошибками (например, переход к несуществующей метке); неверными исходными данными (например, деление на нуль) и т. д.

АДРЕС — число, используемое для указания определенного места в программе (адрес перехода) или в оперативном запоминающем устройстве (адрес ячейки памяти).

АЛГОРИТМ — упорядоченный набор инструкций для решения поставленной задачи за конечное число действий, приводящей к однозначному результату.

«ВШИТЫЙ» — прилагательное, характеризующее математическое обеспечение ЭВМ, которое хранится в постоянном запоминающем устройстве и не утрачивается при отключении электропитания.

ЗАГРУЗИТЬ — ввести программу или исходные данные с внешнего запоминающего устройства во внутреннее.

ЗАЦИКЛИВАНИЕ — состояние ЭВМ, работающей по дефектной программе, когда ни разу не выполняется условие окончания цикла и его повторение может длиться неограниченно долго.

«ЗАЦИКЛИТЬСЯ» — перейти в состояние ЗАЦИКЛИВАНИЯ.

«ЗАЩИТА ОТ ДУРАКА» — средства, препятствующие вводу неверной информации в ЭВМ.

КИЛОБАЙТ (буквально — тысяча байт, сокращенно Кбайт) — $2^{10} = 1024$ байта. Число 1024 иногда называют «двоичной тысячью».

КОМАНДА — оператор Бейсика, выполняемый в непосредственном режиме.

ЛИСТИНГ — текст программы, появляющийся на экране дисплея или на бумаге принтера после выполнения команды LIST.

МЕТКА — знак, используемый в программе для указания определенного места в ней. В большинстве версий Бейсика меткой служит номер строки.

НЕПОСРЕДСТВЕННЫЙ РЕЖИМ — режим работы на ЭВМ, когда операторы Бейсика не формируются в программные строки, а выполняются непосредственно после нажатия клавиши ВК.

«ОБНУЛИТЬ» переменную — придать ей нулевое значение.

ОПТИМИЗАЦИЯ программы — процесс ее совершенствования, направленный на ускорение счета, экономию памяти, повышение точности и т. д.

ОСТАНОВ — прекращение выполнения программы.

ОТЛАДКА ПРОГРАММЫ — устранение ошибок в ней.

ПЕРЕДАЧА УПРАВЛЕНИЯ — изменение последовательности выполнения операторов в том порядке, в котором они записаны в программе.

ПЕРЕХОД — передача управления на строку, указанную в операторе условного или безусловного перехода.

ПОЛЬЗОВАТЕЛЬ — лицо, использующее ЭВМ.

ПРИНЦИП УМОЛЧАНИЯ — правило, которым руководствуется ЭВМ при недостатке исходной информации.

ПРОГОНКА программы — запуск программы, ввод затребованных ею данных и распечатка результата.

«ПРОКРУТИТЬ ЦИКЛ» — один или несколько раз проследить выполнение цикла при прогонке программы или при разработке ее текста.

ПРОТОКОЛ — информация, отображаемая на экране дисплея или на бумаге принтера при выполнении программы.

СБОЙ — прерывание выполнения программы, вызванное содержащейся в ней ошибкой либо неисправностью вычислительной системы.

ЯЧЕЙКА ПАМЯТИ — участок памяти ЭВМ, хранящий 1 байт информации.

ЛИТЕРАТУРА

1. Вуд А. Микропроцессоры в вопросах и ответах. — М.: Энергоатомиздат, 1985. — 184 с.
2. Гарднер М. Математические новеллы. — М.: Мир, 1974. — 456 с.
3. Дьяконов В. П. Справочник по алгоритмам и программам на Бейсике для персональных ЭВМ. — М.: Наука, 1987. — 240 с.
4. Кетков Ю. Л. Программирование на Бейсике. — М.: Статистика, 1978. — 158 с.
5. Кибернетика. Микрокалькуляторы в играх и задачах. — М.: Наука, 1986. — 160 с.
6. Компьютеры: Справочное руководство в 3-х томах / Под ред. Г. Хелмса. — М.: Мир, 1986.
7. Маркелова Л. Н. Эксплуатация программно-управляемой вычислительной машины «Искра 226». — М.: Машиностроение, 1987. — 224 с.
8. Персональные компьютеры. Информатика для всех. — М.: Наука, 1987. — 149 с.
9. Программирование на микроЭВМ «Искра 226» / В. Э. Баласанян, С. В. Богдюкевич, В. А. Шахвердов и др. — М.: Финансы и статистика, 1987. — 264 с.
10. Программирование на языке Бейсик-плюс для СМ-4 / В. П. Семик, Б. Р. Монцибович, Д. П. Непочатых и др. — М.: Финансы и статистика, 1982. — 246 с.
11. Трейстер Р. Персональный компьютер фирмы ИБМ. — М.: Мир, 1986. — 208 с.
12. Трохименко Я. К., Любич Ф. Д. Микрокалькулятор, ваш ход! — М.: Радио и связь, 1985. — 224 с.
13. Уорт Т. Программирование на языке Бейсик. — М.: Машиностроение, 1982. — 278 с.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Аварийный останов 34, 38, 49
Адрес перехода 13
Адрес ячейки 16
Алгоритм 14, 36, 77
Альтернатива 77, 78, 80
Альтернативный вопрос 90
Анализатор ошибок 107, 137

Байт 23
Банк данных 140, 152
Библиотека программ 50, 55
Бит 15
Блок 11, 70, 147
Блок-схема 9, 10

Ввод 11, 126
Версия Бейсика 34, 47, 49, 101, 134, 164
Вещественная переменная 126
Вложение циклов 33, 77
Внешняя память 129, 154
Внутренняя память см. Оперативное запоминающее устройство
Встроенные функции 23
Выбор 77
Вывод 11, 20
Выражение 9
Выход из цикла 28, 42, 48, 49

Генератор псевдослучайных чисел см. Датчик псевдослучайных чисел
Глобальная переменная 114

Данные 71
Датчик псевдослучайных чисел 45, 58
Двумерный массив 65, 83, 95
Диалект Бейсика см. Версия Бейсика
Диалог 14
Диск магнитный 129
Дисковод 130
Дисплей 6, 7
Долговременная память см. Внешняя память

Заголовок цикла 27, 49
«Загрузить» 131
Задержка вывода 47
«Зациклиться» 38
«Защита от дурака» 38
Защита от записи 130
Зона каталога 135
Зона файлов 135

Идентификатор см. Имя
Имя 42
Индекс 44, 47, 70
Индексная переменная 44

Килобайт 129
Клавиатура 6, 7
«Клоп» см. «Солнышко»
Ключевое слово 154, 158, 159
Код символа 103
Команда 54
Комментарий 126
Конкатенация 95
Курсор 8

Листинг 54
Литерная константа 92
Литерная переменная 93
Литерный массив 93
Локальная переменная 114

Массив 44, 46, 47, 50, 64, 70
Математическое обеспечение 54
Матрица 87
Матричный оператор 87, 88
Меню 113, 142, 155
Метка 48
Метод «пузырька» 103
Модем 152

Накопитель см. Внешняя память
«Непосредственный» режим 54
Нестандартные функции 24

«**О**бнулить» 47
Оперативное запоминающее устройство 129, 154
Оперативная память см. Оперативное запоминающее устройство
Оператор 8
Опрос клавиатуры 58, 121
Оптимизация программы 40, 121
Останов 61

Пакет прикладных программ 55
Параметр цикла 27
Передача управления 9, 48
Переменная 18
Переход безусловный 9, 48
Переход по программной ошибке 132
Переход условный 13, 19, 48

Повторение 77
Подпрограмма 74, 80, 113
Пользователь 14
Признак 29, 51, 74
Принтер 70, 117
Принцип умолчания 29, 30, 33
Присваивание 9, 71
Пробел 67
Прогонка программы 39
Программа 8
Программирование сверху вниз 48, 114
Программирование снизу вверх 115
«Прокрутить цикл» 69
Простая переменная 44
Протокол 39
Процедура 113
Псевдослучайное число 45

Распечатка см. Листинг
Редактирование программы 54
Режим реального времени 121
Ремарка 28, 30

Сбой 38, 49
Сектор 131, 135
Сеть ЭВМ 152
Символьная переменная см. Литерная переменная
Системная переменная 48
Склеивание см. Конкатенация
Сложение см. Конкатенация
Следование 77, 78
Служебное слово 8
«Солнышко» 92
Список данных 71
Стандартная программа 55
Стринговая переменная см. Литерная переменная
Строковая переменная см. Литерная переменная
Структурная диаграмма 11, 78, 79
Структурное программирование 29, 30, 48, 77, 148
Счетчик 144

Таймер компьютера 60, 121
Текстовая переменная см. Литерная переменная
Тело цикла 27, 49
Терминал 15
Транслятор 30

Управляющие конструкции 77, 80
Условие 11, 78, 79

Файл 128, 136
Формальный параметр 45
Формат печати 67
Функции пользователя 24

Целочисленная переменная 126
Цикл 27
Цикл «до» 77, 78
Цикл «пока» 77, 78, 80
Цикл с параметром 42, 49, 77

Ячейка памяти 16
Ядро языка 159

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

ГЛАВА 1
ПРОСТЫЕ ПЕРЕМЕННЫЕ

Этюд 1. УГАДАЙ ЧИСЛО 6
Диалоговый режим (14). Бит (15). Адрес ячейки (16).

Этюд 2. САМАЯ ПОПУЛЯРНАЯ ПРОГРАММА 17
Байт (23). Символы Бейсика (23). Знаки препинания в программах (24).

Этюд 3. О РЫБАКАХ И РЫБКЕ 26
О структурном программировании и принципе умолчания (30). Ремарки в программах (30). Транслятор (30).

Этюд 4. КОМПЬЮТЕРНЫЙ РЕБУС 31
Как отсеять лишние решения? (33). «Снисходительность» Бейсика (39).

Этюд 5. «НАЧАЛЬНИК» ПРОГРАММЫ 35
Как упростить программу? (33). Жаргон программистов (39).

Этюд 6. ТИШЕ ЕДЕШЬ — ДАЛЬШЕ БУДЕШЬ 40
Имя переменной (42). Цикл с параметром (42).

ГЛАВА 2
ИНДЕКСНЫЕ ПЕРЕМЕННЫЕ

Этюд 7. ЭВМ БРОСАЕТ ЖРЕБИЙ 43
Массивы в разных диалектах Бейсика (47). Системные переменные (48). Досрочный выход из цикла (48).

Этюд 8. ТРОЙНОЙ ВЫИГРЫШ 50
Редактирование программ (54). Математическое обеспечение ЭВМ (54). Почему не работает программа? (55).

Этюд 9. УЧЕНИК И КОМПЬЮТЕР 56
Почему учтены не все числа? (60). Таймер компьютера (60). STOP и END (61).

Этюд 10. ЕСЛИ БЫ Я РАБОТАЛ КАССИРОМ В КИНОТЕАТРЕ 62
Программные блоки (70). Размерность массивов (70). Списки данных (71).

Этюд 11. СКВОЗЬ ТУМАН К ВЕРШИНЕ ХОЛМА 72
Принципы структурного программирования (77). Подпрограммы (80). Место для подпрограмм (81).

Этюд 12. МУЗЫКА ЦВЕТНЫХ ТОЧЕК 82
Коды символов (87). Матричные операторы (88).

ГЛАВА 3
ЛИТЕРНЫЕ ПЕРЕМЕННЫЕ

Этюд 13. КОМПЬЮТЕР УЧИТСЯ НА СВОИХ ОШИБКАХ . . 89
Обозначение литерных переменных (95). Как еще можно использовать программу? (95).

Этюд 14. ЕСЛИ БЫ Я РАБОТАЛ КАССИРОМ В БАНКЕ . . . 96
Чтение двоичных чисел (100). Синонимы Бейсика (101).

Этюд 15. КОМПЬЮТЕР ЛИКВИДИРУЕТ СВОЮ НЕГРАМОТНОСТЬ 102
Как увеличить глубину сортировки? (106). «Дружественность» Бейсика (107).

Этюд 16. В КАКОЙ ДЕНЬ НЕДЕЛИ РОДИЛАСЬ ВАША БАБУШКА? 108
Подпрограммы и процедуры (113). Какие программы еще можно составить (115).

Этюд 17. МОЮ СЕКРЕТАРШУ ЗОВУТ «ИСКРА» 116
Как усовершенствовать программу (121). Режим реального времени (121).

Этюд 18. «ХРАНИ МЕНЯ, МОЙ ТАЛИСМАН ...» 122
Комментарии в программе (126). Еще один оператор ввода (126). Применение частотного словаря (127).

ГЛАВА 4
ФАЙЛЫ

Этюд 19. ПЕРВАЯ ВСТРЕЧА С ФАЙЛАМИ 128
Запись файлов на диск (134). Разбиение диска (135).

Этюд 20. ПАМЯТЬ НАПРОКАТ 136
Банки данных (140). Операция с файлами (140).

Этюд 21. ПЯТИМИНУТКИ С «ИСКРОЙ» 141
Блочная структура программ (147). Печать на принтере (147). Блоки программы и блоки компьютера (147).

Этюд 22. К ВАМ НЕОЖИДАННО НАГРЯНУЛИ ГОСТИ . . . 149
Что недосмотрела хозяйка? (151). Файлы-программы и файлы данных (151). Компьютер в семье бытовой техники (152).

Этюд 23. КАК Я ЗАВЕДОВАЛ СКЛАДОМ ХИМРЕАКТИВОВ . 153
Как составить электронный справочник? (159). Две группы операторов (159).

Этюд 24. ПОСЛЕДНЕЕ ЗАДАНИЕ 160

ОПЕРАТОРЫ БЕЙСИКА В РАЗЛИЧНЫХ ВЕРСИЯХ . 163
ПЕРЕЧЕНЬ СЛУЖЕБНЫХ СЛОВ, ИСПОЛЬЗУЕМЫХ В ПРОГРАММАХ КНИГИ 165
СЛОВАРЬ ТЕРМИНОВ 168
ЛИТЕРАТУРА 169
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ 170

Научно-популярное издание

Валерий Федорович Очков,
Юрий Васильевич Пухначёв

24 ЭТЮДА НА БЕЙСИКЕ

Зав. редакцией И. Г. Дмитриева
Редактор Л. А. Табакова
Худож. редактор С. Л. Витте
Мл. редактор Т. А. Студеникина
Техн. редактор Г. А. Полякова
Корректоры Т. М. Васильева и Т. М. Иванова
Оформление художника В. С. Филатовича

ИБ № 2160

Сдано в набор 19.10.87. Подписано в печать 8.06.88. А 11210.
Формат $60\times90^{1}/_{16}$. Бум. офсетная. Гарнитура журн.-рубл. Печать офсетная.
Усл. п. л. 11,0. Усл. кр.-отт. 43,0. Уч.-изд. л. 11,29.
Тираж 50000 экз. Заказ 5664. Цена 1 р. 10 к.

Издательство «Финансы и статистика», 101000, Москва, ул. Чернышевского, 7.
Набранс в типографии им. Котлякова
издательства «Финансы и статистика»
Государственного комитета СССР
по делам издательств, полиграфии и книжной торговли.
195273, Ленинград, ул. Руставели, 13.
Отпечатано в типографии «Коммунист» ЦК КП Азербайджана
370146. г. Баку, Метбуат проспекти, 529-й квартал.

Это шляпа?
ДА
Можно носить на голове?
Это подводная лодка?
ДА
Оно сделано из металла?
НЕТ
Это лошадь?
ДА
Это попугай?
ДА
Может оно скакать?
НЕТ
Это автомобиль?
ДА
У него есть колеса?
НЕТ
Это слон?
ДА
У него есть перья?
НЕТ
У него больше чем
Это акула?
НЕТ
С ним можно говорить?
ДА
У него есть жабры?
ДА
Это кит?
НЕТ
На нем можно ездить?
НЕТ
ДА
Живет оно в воде?
НЕТ
Это кот?

1 р. 10 к.

«Финансы и статистика»
Москва 1988

Большинство пользователей персональных компьютеров будет работать по готовым программам. Но книга представляет интерес и для них: она расскажет о "душе" компьютера — его математическом обеспечении.